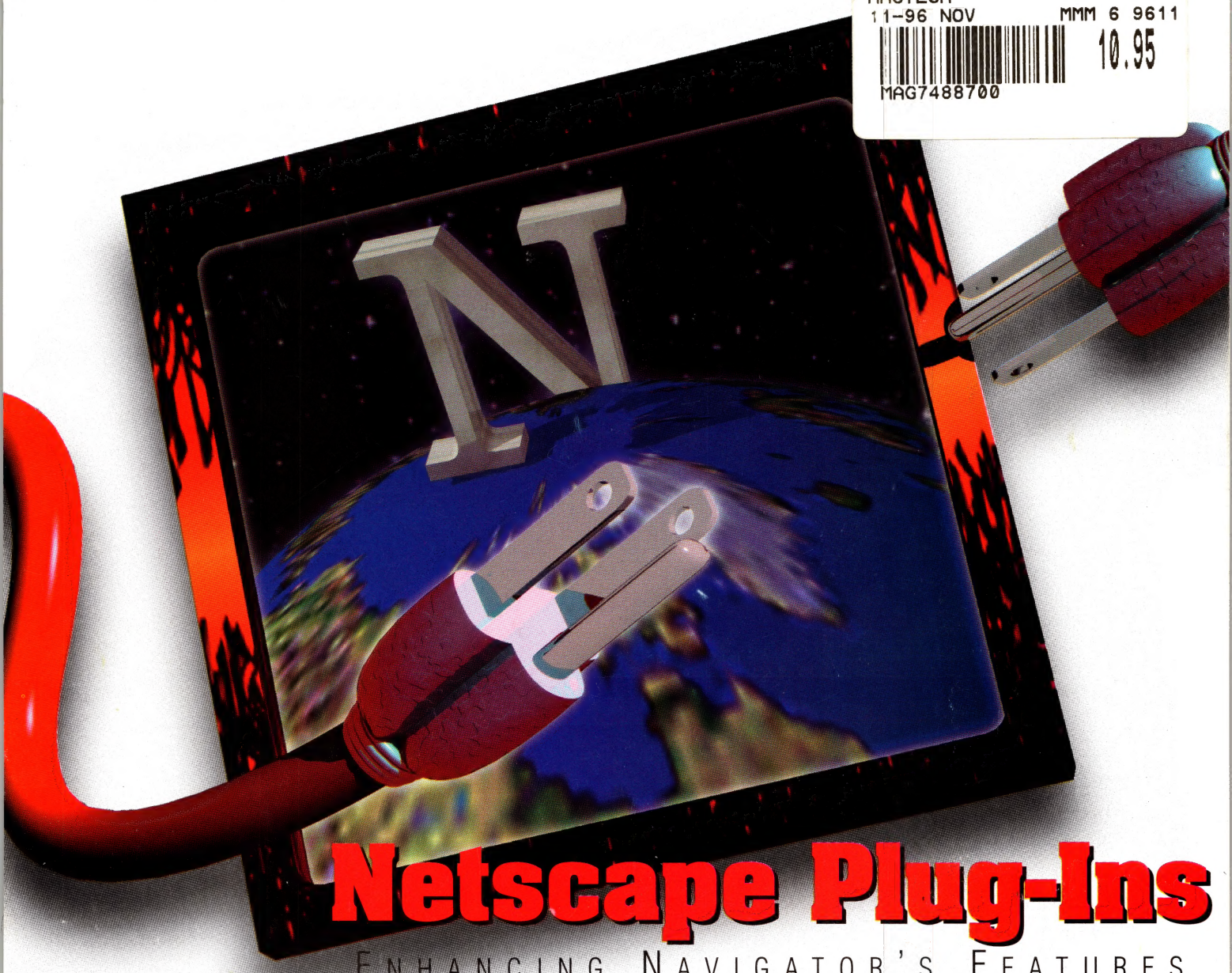
For Macintosh Programmers & Developers

MacTech™ MAGAZINE

INSIDE:
Mac OS 8
Address
Spaces
& Memory

Vol. 12, No. 11 • November 1996



Netscape Plug-Ins

ENHANCING NAVIGATOR'S FEATURES

PLUS

ENABLING YOUR APPS TO DO SMTP

0 32128 74887 8 11

$5.85 US
$6.95 Canada
ISSN 1067-8360
Printed in U.S.A.

Symantec Café.
Where mere mortals turn into overnight Java Gurus.

Java Development Tools For Windows and Macintosh™

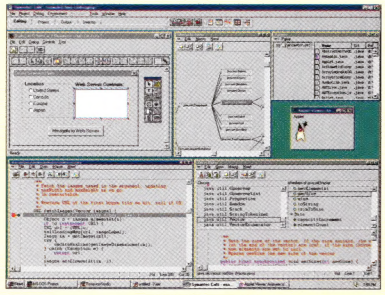
Want to turn your Web site into a dazzling display of Java™ virtuosity? Welcome to Symantec Café.™

Symantec Café is a completely integrated Java development environment that lets you create real-time Java applets in virtually no time at all. Even if you've never worked in Java before.

Symantec Café is packed with full-featured graphical development tools including class and hierarchy browsers. A wickedly fast 32-bit Java compiler. An integrated source-level debugger. And visual tools that let you draw user interface elements like forms and menus directly on screen.

Designed for Microsoft® Windows®95 Mac OS

Naturally, Symantec Café is totally Windows 95 and Windows NT compliant. So grab a copy today and achieve Guru status for yourself. Not to mention raves for your Web sites.



- **Visually create Java forms and menus.**
- **Integrated graphical source-level debugger.**
- **Native compiler builds Java apps up to 13 times faster than Sun's compiler.**
- **Just-in-time compiler makes your Java apps run up to 25 times faster.**
- **Includes *Introduction to Java Programming.***

Symantec Café is available at your local computer store.
For more information call us at 1-800-277-3946, ext. 9H27.
Or visit the Symantec Java Central Web Site at http://cafe.symantec.com.

SYMANTEC.®

Eddy Award Winner** for **Best New Developer Tool
– MacUser Editors Choice Awards, 1993

"A distinct improvement over ResEdit."
– MacTech / MacTutor

"Resorcerer's data template system is amazing!"
– Bill Goodman, author of Compact Pro

"Nuke ResEdit! Resorcerer is mission-critical for us."
– Dave Winer, Userland Frontier

"The color pixel editors are wonderful! A work of art!"
– Dave Winzler, author of Microseeds Redux

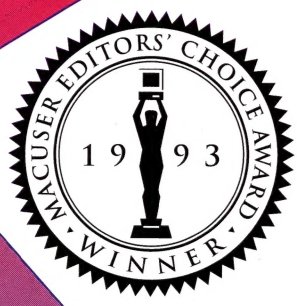
"Every Macintosh developer should own a copy of Resorcerer."
– Leonard Rosenthol, Aladdin Systems

"Resorcerer will pay for itself many times over in saved time and effort."
– MacUser review

"The template that disassembles 'PICT's is awesome!"
– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensible in its own creation!"
– Doug McKenna, author of Resorcerer

"...a wealth of time-saving tools."
MacUser Review, Dec. 1992



RESORCERER®

Version 1.2.4

The Resource Editor for the Macintosh Wizard

ORDERING INFO

Needs: ≥Mac Plus, ≥ Sys 4.2, 1MB
Likes: ≥Mac Plus, ≥ Sys 7.0, 2MB
32-bit clean, AU/X compatible

Price: $256 (decimal)
(Educational, quantity, or other discounts available)

Includes: 500 page manual
60-day Money-Back Guarantee
Domestic UPS ground shipping

Payment: Check, PO's, or Visa/MC

Extras (call us):
COD, FedEx, UPS Blue/Red, International Shipping

Downloadable Demos/Updaters:
AppleLink: Software Sampler
AOL: Software Libs/Development
CompuServe: MACDEV/Tools
or call us.

New 1.2 Features:

- **New 'cicn', 'ppat', 'crsr', 'acur', 'pltt', 'clut' editors**
- **Powerful icon family editing (all 9 icon types)**
- **Color pixel anti-aliasing, dithering, and lots more**
- **Complete 'PICT' disassembly and reassembly**
- **Resource sorting; ROM resource browsing**
- **120 template field parsing types now supported**
- **New insertion & deletion template field types**
- **Text-only 'PICT' resources**
- **Lots of improvements throughout**

- Easier, faster, more Mac-like, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Help, AppleEvent, OCE, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Well-designed, helpful developer tools being added all the time
- Relied on by thousands of Macintosh developers around the world

MATHEMÆSTHETICS, INC.
P.O. Box 298 • Boulder • CO • 80306-0298 • USA
Phone: (303) 440-0707 • Fax: (303) 440-0504
AppleLink/AmericaOnline: RESORCERER • Internet: resorcerer@aol.com

For Macintosh Programmers & Developers

MacTech™ MAGAZINE

A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 800-MACDEV-1

DEPARTMENTS	E-Mail/URL
Orders, Circulation, & Customer Service	cust_service@devdepot.com
Press Releases	press_releases@mactech.com
Ad Sales	ad_sales@mactech.com
Editorial	editorial@mactech.com
Programmer's Challenge	prog_challenge@mactech.com
Online Support	online@mactech.com
Accounting	accounting@mactech.com
Marketing	marketing@mactech.com
General	info@mactech.com
Web Site (articles, info, URLs and more...)	http://www.mactech.com

PRINTED WITH SOY INK™

This publication is printed on paper with recycled content.

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

TABLE OF CONTENTS

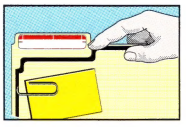
November 1996 • Volume 12, No.11

For Macintosh Programmers & Developers

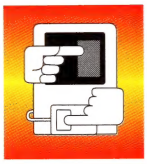
MacTech™ MAGAZINE

FEATURE ARTICLES

REGULAR COLUMNS



GETTING STARTED



By Dave Mark

Two Java Grid Layouts

Last month, we introduced the Java Layout Manager and saw the power of layouts combined with panels. This month, we'll look at two important layout classes, GridLayout and GridBagLayout, and present a series of applets that bring these two classes to life.

There is a newly reformatted set of Java API documentation, collectively known as the 1.0.2 API. If you don't already have this, go get it now. The URL is:

http://java.sun.com/doc/api_documentation.html

There are two Mac download links on this page. Though it's bigger, you might try the .hqx file (as opposed to the .bin file). I'm not sure why, but when I downloaded the .bin file and dropped it on StuffIt Expander, the final .sea file was corrupted. On the other hand, by the time you read this, the problem most likely will have been corrected.

GridLayout

As its name implies, the GridLayout lays out its components in a grid. GridLayout has two constructors:

```
public GridLayout( int  rows, int  cols );
```

This one creates a grid layout with the specified rows and columns. As you'll see, GridLayout does the best it can to lay the current set of components out in this configuration. But what if you have too few components? Or too many? This month's sample applets are ideal for experimenting.

The second constructor adds two new parameters:

```
public GridLayout( int  rows, int  cols,
                   int  hgap, int  vgap );
```

This version creates a grid layout with the specified rows and columns, but also lets you specify a minimal horizontal and vertical gap to appear between the components.

GridLayout is pretty straightforward. Here's a sample applet to take it for a spin.

- Launch the CodeWarrior IDE and create a new "Java Applet" project called GridLayout.µ.
- Create a new source code window, type in the following source code, save as GridLayout.Java, and add to it the project.

```
import java.awt.*;

public class MyGrid extends java.applet.Applet
{
  public MyGrid()
  {
    setLayout( new GridLayout( 4, 4 ) );

    add( new Button( "1" ) );
    add( new Button( "2" ) );
    add( new Button( "3" ) );
    add( new Button( "4" ) );
    add( new Button( "5" ) );
    add( new Button( "6" ) );
    add( new Button( "7" ) );
    add( new Button( "8" ) );
    add( new Button( "9" ) );
    add( new Button( "10" ) );
    add( new Button( "11" ) );
    add( new Button( "12" ) );
    add( new Button( "13" ) );
    add( new Button( "14" ) );
    add( new Button( "15" ) );
    add( new Button( "16" ) );
  }
}
```

- Create a second source code window, type in the following HTML, save as GridLayout.html, and add it to the project as well.

```
<title>GridLayout</title>
<hr>
<applet codebase="GridLayout Files" code="MyGrid.class"
```

Software Developers:

Software Piracy Burns Your Profits.

NSTL Study Rates HASP As Number One!

A recent test conducted by the National Software Testing Labs compared the flagship PC products of four leading software protection vendors. The result? HASP was rated the clear overall winner - and number one in all the major comparison categories. And if the world's leading independent testing lab says HASP is the best, who are we to disagree?

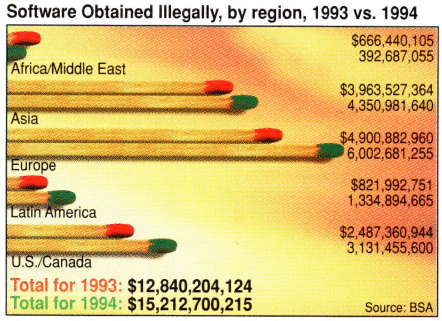
NSTL TEST RESULTS, OCTOBER 1995

Scoring Category	Aladdin HASP	Rainbow Sentinel
Security	9.3	6.3
Ease of Learning	9.1	7.1
Ease of Use	8.3	7.2
Versatility/Features	10	8.7
Compatibility/ Power Consumption	6.7	6.5
Speed of API Calls	0.9	1.2
Final Score	**8.5**	6.5

*For a full copy of the NSTL report, contact your local HASP distributor.

Each year, the illegal use of software consumes nearly 50% of your potential revenues. With the flames of piracy eating away at your profits, can you afford not to protect your software?

Software Obtained Illegally, by region, 1993 vs. 1994



MacHASP® is widely acclaimed as the world's most advanced software protection solution for Mac and Power Mac computers. Since 1984, thousands of leading Mac and PC developers have used over two million MacHASP and HASP keys to protect billions of dollars worth of software. Why? Because MacHASP's security, reliability, and ease-of-use led them to a simple conclusion: MacHASP is the most effective software protection system available.

Today, more software developers are choosing MacHASP than any other software protection method. To learn why, and to see how easily you can increase your revenues, call today to order your low-cost MacHASP Developer's Kit.

1-800-223-4277

www.aks.com

ALADDIN™

The Professional's Choice

North America — **Aladdin Knowledge Systems Inc.**
Tel: (800) 223 4277, 212-564 5678
Fax: 212-564 3377, E-mail: hasp.sales@us.aks.com

Int'l Office — **Aladdin Knowledge Systems Ltd.**
Tel: +972-3-636 2222, Fax: +972-3-537 5796
E-mail: hasp.sales@aks.com

Germany — **FAST Software Security AG**
Tel: +49 89 89 42 21-37, Fax: +49 89 89 42 21-40
E-mail: info@fast-ag.de

United Kingdom — **Aladdin Knowledge Systems UK Ltd.**
Tel: +44 1753-622266, Fax: +44 1753-622262
E-mail: sales@aldn.co.uk

Japan — **Aladdin Japan Co., Ltd.**
Tel: +81 426-60 7191, Fax: +81 426-60 7194
E-mail: sales@aladdin.co.jp

Benelux — **Aladdin Software Security Benelux B.V.**
Tel: +31 24-641 9777, Fax: +31 24-645 1981
E-mail: 100526.1356@compuserve.com

yes It runs with NetWare Developer tested only

Available and Ready for OS/2

Microsoft Windows™ Compatible

Mac™ OS

■ **Aladdin Russia** 095 9230588 ■ **Australia** Conlab 03 98985685 ■ **Chile** Micrologica 02 2221388 ■ **China** Shanghai LIRI 021 64377828 ■ **Czech** Atlas 02 766085 ■ **Denmark** Berendsen 039 577316 ■ **Egypt** Zeineldein 02 3604632 ■ **Finland** ID-Systems 0 8703520 ■ **France** 1 40859885 ■ **Greece** ■ Unibrain 01 6756320 ■ **Hong Kong** Hastings 02 5484629 ■ **India** Solution 011 2148254 ■ **Italy** Partner Data 02 26147380 ■ **Korea** Dae-A 02 8484481 ■ **Mexico** SiSoft 5 2087472 ■ **New Zealand** Training 04 5666014 ■ **Poland** Systherm 061 480273 ■ **Portugal** Futurmatica 01 4116269 ■ **Romania** Ro Interactive 064 140283 ■ **Singapore** ITR 065 5666788 ■ **South Africa** D Le Roux 011 8864704 ■ **Spain** PC Hardware 03 4493193 ■ **Switzerland** Opag 061 7169222 ■ **Taiwan** Teco 02 5559676 ■ **Turkey** Mikrobeta 0312 4670635 ■ **Yugoslavia** Asys 021 623920

See Us at Comdex Booth S7130

```
width=200 height=200>
</applet>
<hr>
<a href="GridLayout.java">The source.</a>
```

- Remove the two <replace me> files from the project (⌘-click on the files to select them, then hit option-delete).
- Edit the project prefs, specifically, the Java Project pane. Set the Project Type popup to "Class Folder" and type "GridLayout Files" (without the quotes) as the Folder Name.

It's important that the Class Folder preference *exactly* match the "codebase" attribute in your HTML file.

- Once all your source is in place, select **Make** from the **Project** menu to generate your class file.
- To run your applet, drop the html file onto your Java Applet runner or Java-capable browser. Figure 1 shows my version running in a Netscape window.

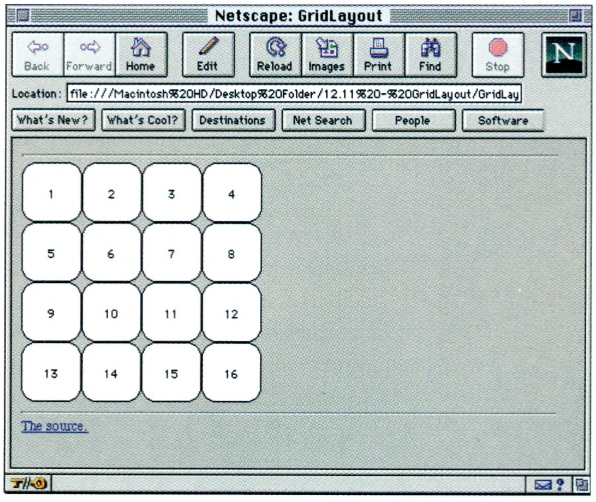


Figure 1. The GridLayout applet, running in Netscape.

The GridLayout Source Code

Here's how the source code works. First comes the normal opening stuff, the import statement and class definition. The setLayout() statement creates a new GridLayout object with 4 rows and 4 columns, and makes it the current layout.

```
import java.awt.*;

public class MyGrid extends java.applet.Applet
{
  public MyGrid()
  {
    setLayout( new GridLayout( 4, 4 ) );
```

Next, we create a series of 16 buttons and add them to the current frame.

```
    add( new Button( "1" ) );
    add( new Button( "2" ) );
    add( new Button( "3" ) );
    add( new Button( "4" ) );
    add( new Button( "5" ) );
    add( new Button( "6" ) );
    add( new Button( "7" ) );
    add( new Button( "8" ) );
    add( new Button( "9" ) );
    add( new Button( "10" ) );
    add( new Button( "11" ) );
    add( new Button( "12" ) );
    add( new Button( "13" ) );
    add( new Button( "14" ) );
    add( new Button( "15" ) );
    add( new Button( "16" ) );
  }
}
```

That's it! When you run the applet, your 16 buttons will appear in a 4 by 4 grid. The width and height of the buttons is determined by the width and height attributes in your HTML's applet tag. Make the applet frame wider, the buttons will each be made wider. Make the frame taller, the buttons will each be made taller.

You can also affect the results by changing the parameters you pass to the GridLayout constructor. Experiment.

GridLayout, Version 2

Here's another GridLayout applet. This one uses all four constructor parameters, and includes a nifty little trick you'll want to remember. First, here's the code:

```
import java.awt.*;

public class MyGrid extends java.applet.Applet
{
  int    numButtons;
  String att;

  public void init()
  {
    att = getParameter( "NUMBUTTONS" );
    numButtons = Integer.valueOf(att).intValue();

    setLayout( new GridLayout( 2, 20, 5, 20 ) );

    for ( int i=1; i<=numButtons; i++ )
      add( new Button( ""+i ) );
  }
}
```

Next, here's the HTML:

```
<title>GridLayout</title>
<hr>
<applet codebase="GridLayout Files" code="MyGrid.class"
width=400 height=200>
<param name="NUMBUTTONS" value="16">
</applet>
<hr>
<a href="GridLayout.java">The source.</a>
```

Figure 2 shows the applet in action, running under Netscape. Let's take a look at this source.

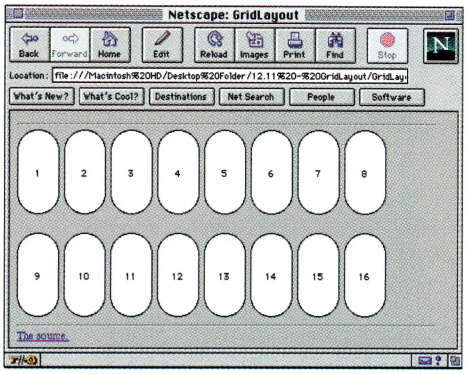


Figure 2. Another GridLayout applet using all 4 parameters.

GridLayout 2 Source Code

This version of GridLayout.Java starts off in the same way, but does its creating in init() instead of in MyGrid(). This gives us access to the HTML parameters. I'm not sure why getParameter() doesn't work from within MyGrid(), but I'll look into it.

```
import java.awt.*;

public class MyGrid extends java.applet.Applet
{
  int    numButtons;
  String att;
```

If you look back at the HTML, you'll see that we stuck in a parameter with the name "NUMBUTTONS" and a value of "16". We call getParameter() to pick up the parameter and Integer.valueOf(att).intValue() to convert the returned string to a number. Next, we create a new GridLayout using all 4 parameters and make it the current layout. Note that we've specified 2 rows and 20 columns, with 5 pixels horizontally and 20 pixels vertically between components. The Layout Manager uses the row value first for GridLayouts, so the fact that you've specified 20 columns really has no affect. Try using 0 for a column value.

```
  public void init()
  {
    att = getParameter( "NUMBUTTONS" );
    numButtons = Integer.valueOf(att).intValue();

    setLayout( new GridLayout( 2, 20, 5, 20 ) );
```

Now for the cool trick. In our earlier example, we explicitly specified the name of each button using Strings like "1", "2", etc. In this case, we add the loop counter, i, to the null string to produce a string representation of the loop counter. Basically, we've forced Java to do the typecasting from number to String for us, since the + operator is expecting a String on both sides. Pretty cool, eh?

```
    for ( int i=1; i<=numButtons; i++ )
      add( new Button( ""+i ) );
  }
}
```

The GridBagLayout

The GridLayout works pretty well if all your components are the same size. But, suppose you are working with all sorts of elements; some tall, some wide, whatever. In this case, the GridLayout won't work particularly well (it'll waste a lot of screen real estate). Fortunately, there is a complex, grid-based class designed to handle variable sized components.

GridBagLayout and its sister class, GridBagConstraints, allow you to customize a layout that allows components to span multiple grid cells. The GridBagConstraints class features a number of variables, each designed to constrain any components added to the current GridBagLayout. Take a look at the GridBagConstraints class declaration:

```
public  class  Java.awt.GridBagConstraints
    extends  Java.lang.Object
    implements Java.lang.Cloneable
{
    // Fields
    public int anchor;
    public int fill;
    public int gridheight;
    public int gridwidth;
    public int gridx;
    public int gridy;
    public Insets insets;
    public int ipadx;
    public int ipady;
    public double weightx;
    public double weighty;

    // the anchor field has one of the following values
    public final static int CENTER;
    public final static int EAST;
    public final static int NORTH;
    public final static int NORTHEAST;
    public final static int NORTHWEST;
    public final static int SOUTH;
    public final static int SOUTHEAST;
    public final static int SOUTHWEST;
    public final static int WEST;

    // the fill field has one of the following values
    public final static int BOTH;
    public final static int HORIZONTAL;
    public final static int NONE;
    public final static int VERTICAL;

    // default value for gridheight, gridwidth
    public final static int REMAINDER;

    // default value for gridx, gridy
    public final static int RELATIVE;

    // Constructors
    public GridBagConstraints();

    // Methods
    public Object clone();
}
```

To use a GridBagLayout, you'll create a GridBagLayout object along with a corresponding GridBagConstraints object, then make the GridBagLayout the current object. Next, you'll set your GridBagConstraints fields to the settings you prefer. Now you are ready to start adding components to the current frame. All the added components will be formatted according to the current GridBagConstraints settings. Change the constraints settings and add some more components. The changed

constraints only affect future components, not the components that were already added.

anchor determines where, within a cell, the component is placed. **fill** determines if the component is reissued to fill its cell and, if so, how. **gridheight** specifies the number of cells in a column. **gridwidth** specifies the number of cells in a row. REMAINDER is used to mark a component as the last in its row or column. RELATIVE is used to mark a component as next to last.

gridx and **gridy** allow you to specify where to place the component in the grid. A value of (0,0) will put the next component in the upper left corner. A value of RELATIVE will put the component either at the end of a row (in the case of gridx) or column (in the case of gridy).

insets specifies the number of pixels of padding on any side of a cell. **ipadx** and **ipady** allow you to specify the padding in pixels within a cell.

Finally, **weightx** and **weighty** allow you to specify how much horizontal and vertical space this component should consume when the available extra display area is divvied up between all the components in a row or column.

A GridBagLayout Example

There is really no way to truly appreciate the GridBagLayout without playing with an example. The following is one of the standard Sun applets, stripped down to make it as small as possible. Take some time to play with this applet. Change the constraints, experiment with all the fields and settings to see what they do.

Here's the source code:

```
import java.awt.*;

public class MyGridBag extends java.applet.Applet
{
  public MyGridBag()
  {
    GridBagLayout   gridBag = new GridBagLayout();
    GridBagConstraints  constraints =
                new GridBagConstraints();

    setLayout( gridBag );

    constraints.fill  =  constraints.BOTH;
    constraints.weightx  =  1.0;

    ConstrainedButton("Button1",  gridBag,  constraints );
    ConstrainedButton("Button2",  gridBag,  constraints );
    ConstrainedButton("Button3",  gridBag,  constraints );

    constraints.gridwidth  =  constraints.REMAINDER;

    ConstrainedButton("Button4",  gridBag,  constraints );

    constraints.weightx  =  0.0;

    ConstrainedButton("Button5",  gridBag,  constraints );

    constraints.gridwidth  =  constraints.RELATIVE;

    ConstrainedButton("Button6",  gridBag,  constraints );

    constraints.gridwidth  =  constraints.REMAINDER;

    ConstrainedButton("Button7",  gridBag,  constraints );

    constraints.gridwidth  =  1;
    constraints.gridheight  =  2;
    constraints.weighty  =  1.0;

    ConstrainedButton("Button8",  gridBag,  constraints );

    constraints.weighty  =  0.0;
    constraints.gridwidth  =  constraints.REMAINDER;
    constraints.gridheight  =  1;

    ConstrainedButton("Button9",  gridBag,  constraints );
    ConstrainedButton("Button10",  gridBag,  constraints );
  }

  void ConstrainedButton( String title,
    GridBagLayout layout, GridBagConstraints constraints )
  {
    Button button = new Button( title );
    layout.setConstraints( button, constraints );
    add( button );
  }
}
```

Here's the HTML:

```
<title>GridBagLayout</title>
<hr>
<applet codebase="GridBagLayout Files" code="MyGridBag.class"
width=400 height=100>
</applet>
<hr>
<a href="GridBagLayout.java">The source.</a>
```

Figure 3 shows the results of this applet, when run in Netscape.

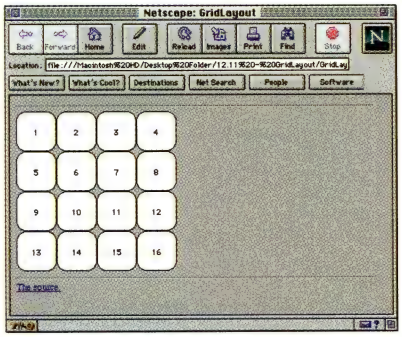


Figure 3. The classic GridBagLayout applet from Sun.

Till Next Month...

As you go through the GridBagLayout source, pay attention to the use of REMAINDER and RELATIVE. Remember, you are marking a component as 2nd to last and last in its row or column. For example, Button4 should be the last in its row. Button6 should be RELATIVE (2nd to last) while Button7 should be REMAINDER (last). All the buttons should use a gridheight of 1 except for Button8, which will use a gridheight of 2. You get the idea.

Next month, we'll take a look at double-buffered animation, something that Java makes fairly easy to do. Till then, have a Happy Thanksgiving and save me a wishbone...

FROM THE FACTORY FLOOR

By Dave Mark, Metrowerks

A Brand New Constructor

This month's Factory Floor interview is with Eric Scouten, Robin Mair, and Clint Popetz, three members of Metrowerks' Constructor team. In case you've never used it before, Constructor is the visual front end that makes it easy to design interfaces for your Macintosh and Java applications and applets. With the release of CodeWarrior 10 in September, Constructor sports a new look and offers some new functionality. Read on to find out more.

Dave: How would you compare the old (CW8) and new (CW10) versions of Constructor?

Eric: There are now two Constructors: one for PowerPlant (MacOS) and one for Java. The Java Constructor debuted on CW10 and brings the same interface-building capabilities to the Java language framework (AWT) that we've been providing for our C++ framework for two years now.

In the PowerPlant Constructor, we've added several new resource editors since CW8. In CW9, we added the ability to edit menu bars. It's pretty slick. The menu bar editor relies heavily on the Mac's drag-and-drop interface. You can drag menu items around from one menu to another in a single mouse action. In CW10, we added support for many of the bitmap resource formats that people use in MacOS applications. You can edit icon suites, PICT resources, pattern resources, and more.

The other changes you'll notice are not features, per se, but improvements in the user experience. In CW9, we began adopting the Apple Grayscale Appearance; this look is now almost complete in CW10, and is also visible in other parts of the CW product line (especially the IDE). Also, the CW8 Constructor had a lot of problems with stability. We've tightened up the code quality significantly since then.

Dave: Why are there two separate Constructors for Java (the AWT) and MacOS (for PowerPlant)?

Eric: There were two things that motivated this decision: code stability and code size. Constructor for Java contains a lot of new technology, such as an embedded copy of the Java virtual machine. This introduces some additional system requirements (such as the Code Fragment Manager for 68K) and adds quite a bit to the size of the final application. The Java runtime overhead is unnecessary in the Constructor for MacOS.

I also wanted to make sure that the development of Constructor for Java did not, in any way, disrupt the stability of the PowerPlant Constructor. Since the Java version was finalized fairly late in the CW10 release cycle, this was a significant concern.

In the future, both versions will be built from the same source files and will share about 80% common code. However, I expect that they will always remain separate. The project models are different, and the products will be growing in different directions.

Clint: As Eric said, embedding the Java runtime in Constructor creates a noticeable increase in memory footprint and binary size. We didn't want to foist this on unsuspecting Constructor users. In addition, we felt that the intersection of Java programmers and PowerPlant programmers is pretty small at this point, so keeping the products separate seemed like a good move.

Dave: What is the Code Generation model for Java Constructor?

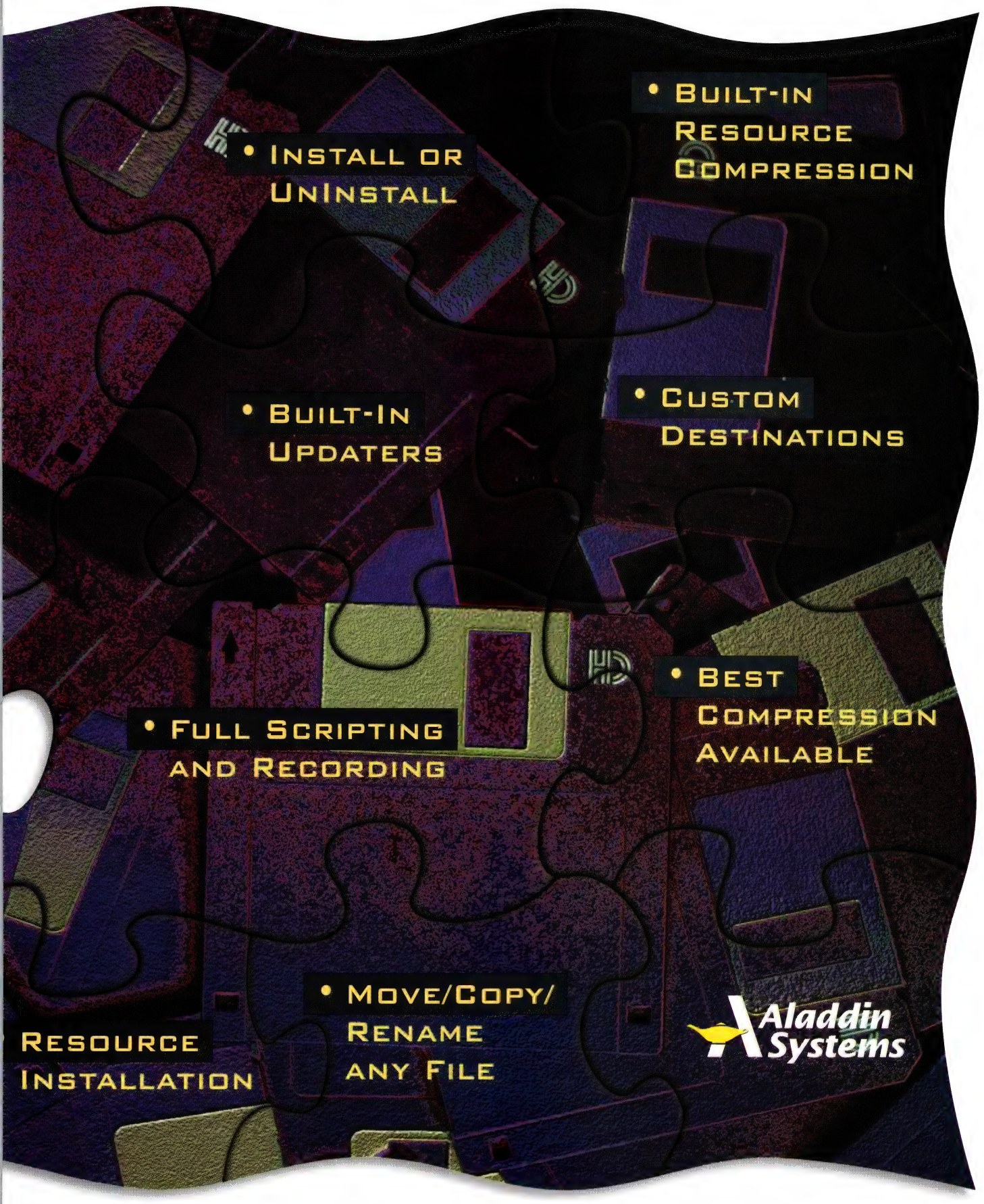
Our competitors' habit of charging for each piece of their installer program is a bit puzzling.

- CREATE UP TO 128 CUSTOM INSTALLS
- SUPPORTS LIVE OBJECTS INSTALLATION

The reason we call StuffIt InstallerMaker 3.1 the Complete Installation Solution is because we include **everything you need** to prepare a professional package of files for installation on your user's machines. An installer, updater, and an uninstaller are just three components included for the same price one of our fine competitors charges for just the installer. Using our installer reduces technical support calls due to end-user errors during installation, saves disks (if distributing on floppies) or download time (if distributing on-line). You will save more than the cost of the Installer license by significantly reducing distribution costs. And the puzzle will be solved. You'll have every piece in place.

Download a **free,** fully-functional copy of StuffIt InstallerMaker 3.1 from **www.aladdinsys.com**, or call (408) 761-6200 and ask for Developer Sales.

STUFFIT INSTALLERMAKER 3.1


• Built-in Resource Compression
• Install or UnInstall
• Custom Destinations
• Built-In Updaters
• Best Compression Available
• Full Scripting and Recording
• Move/Copy/ Rename any File
Resource Installation
Aladdin Systems

Eric: It's pretty transparent, really. When you save a Constructor file, a Java source file is generated at the same time. It contains all the necessary code to rebuild the interface that you've described in the visual editor.

Clint: The code generation mechanism (which was written completely in Java) is meant to duplicate as closely as possible the UReanimator functionality in PowerPlant. So, the generated code consists of a series of classes, one for each component in the hierarchy, which know how to reanimate the target component. This source is not meant to be modified, since it is not the actual component subclass source. The developer can create their own Component subclasses in separate files, and then invoke the Reanimator methods to build the interface at runtime. Code to do this might look like:

```
MyFrame theFrame =
(MyFrame)Reanimator.Reanimate("MyFrame","theFrameName");
```

In addition, the developer may wish to grab references to components in the newly created hierarchy. This functionality, which PowerPlant users will recognize as similar to the FindPaneByID functionality, is also provided by the Reanimator. Sample code to locate a TextArea in theFrame might look like:

```
TextArea theArea =
(TextArea)Reanimator.Locate("TextArea","theArea",theFrame);
```

Dave: Over time, more and more resource editing appears to be moving into Constructor. Will Constructor eventually be able to take advantage of TMPLs? How much resource editing do you see moving into Constructor?

Eric: Right. Constructor began its life as a special-purpose resource editor, which knew about only the PowerPlant-specific resources (PPob and Txtr). We've gradually been adding more generic resource editing capabilities to it: menus, icons, etc.

These features are part of a long-term plan to evolve Constructor into a general-purpose resource editor and application builder. Editing resources described by TMPLs is an essential part of this plan, and it should be supported fairly soon (in either CW11 or CW12). We are also designing a plug-in API for resource editors, so you can write your own editors for custom resource types.

Dave: What Apple events does Constructor support now (in CW10)? What about new Apple events for future releases?

Eric: Apple event support in Constructor is pretty minimal. We've found that users view Constructor as an interactive program, and don't typically want to script the process of designing a visual interface.

In the future, Constructor may have some Apple event interactions with the CodeWarrior IDE, to support building and running applications directly from Constructor.

Dave: Constructor allows you to design a CPPb resource that describes a "custom display class", that is, a class derived from an existing PowerPlant display class. How does this mechanism work?

Eric: The properties (location, size, color, etc.) for each pane or view in a layout are stored in a bytestream. Constructor has built-in knowledge about all of the classes that are part of the PowerPlant framework itself.

Invariably, developers need to create their own subclasses of the PowerPlant building blocks to display the content that is specific to their application. (An example might be to create a subclass of LTableView that displays the list of message titles, senders, etc. in an e-mail application.)

The custom display class mechanism allows you to use Constructor to describe these specialized classes. If the class requires extra data to be in the bytestream, you create a CPPb resource to tell Constructor what these properties are. These properties then appear in the property inspector alongside the properties for the built-in class that you've derived from.

Custom display classes are due for a major overhaul in the next release of Constructor. We'll be making the process quite a lot easier and less error-prone.

Dave: Products such as WebBurst allow you to build a complete applet or application using Constructor-like techniques, while Constructor has focused more on the specification of the user interface. Will Constructor be moving more into the application building model?

Robin: Constructor really started as a tool that provided easier access to the capabilities of PowerPlant and its pane/view mechanism, allowing graphical manipulation of the pane hierarchies. It is certainly our intent to move Constructor forward into the application building realm in an effort to make it easier for our customers to build applications. We will start moving in this direction by allowing the user to handle many of the more mundane aspects of application building directly from within Constructor. This would include providing tools for editing the standard resources such as, 'BNDL', 'FREF', etc. The inclusion of the new icon editor will also facilitate this by allowing the icons for the application to be created within Constructor. We don't necessarily want to become the next ResEdit, but we want to

Announcing

MICROGUARD PLUS.™

Why so many developers are switching to MicroGuard copy protection

MicroGuard is committed to uncompromising technological superiority

"Technology at its peak" is our commitment to you. That is why we have brought you **MicroGuard Plus**™. And, MicroGuard Plus is **100% backwards** compatible with MicroGuard. This means MicroGuard and MicroGuard Plus can be used interchangeably. Just as we promised!

MicroGuard has now surpassed its own technological lead, actually improving on the best

MicroGuard Plus is everything MicroGuard is, plus 40-bit encryption, two additional passwords, 64-Bit Array, 32-byte public area, 45% size reduction, enhanced counter and more. In addition, MicroGuard Plus offers two new utilities: **QuickGuard**™ and EasyGuard™ allow you to protect your applications without touching your source code. The only feature that is not plus is the price. :-)

MicroGuard offers you the most sophisticated network protection

Our network protection, **MicroGuard Net**™, is so superior, we had to hire an Apple network engineer to execute our specifications.

MicroGuard is the best selling Macintosh key in the world

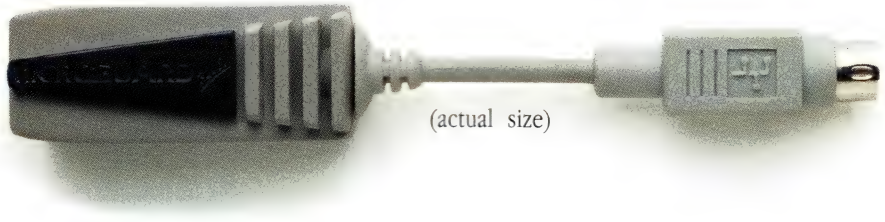
MicroGuard sells more Macintosh copy-protection keys than anyone else in the world!

MicroGuard is the only key developed by Mac developers, and is the first and only 100% ADB savvy key

We have been developing Mac applications as a seed development house since 1984. We are not a PC protection company that has come to you with a Mac product. MicroGuard is fully ADB savvy and offers extended addressing. Unlike other protection devices, MicroGuard never clashes with other keys. Only MicroGuard offers this level of sophistication.

MicroGuard delivers developer support within 24 hours

We will answer any inquiry you have within 24-hours. We also have a fully loaded AppleLink bulletin board which contains all our libraries, tech notes, Q&A and nearly everything you'll ever need!



(actual size)

For more information and to order a Developer's Kit or to receive a free CD ROM about MicroGuard, please contact us at:

MicroGuard USA:	Tel: (303) 320-1628	• Fax: (303) 320-1599	• AppleLink: M.GUARD
International:	Tel: (972) 3 558-2345	• Fax: (972) 3 558-2344	• AppleLink: MICROGUARD

Mac™OS

Pirate Busters

enable our users to edit the resources commonly associated with building an application on the Mac.

In addition, we want to be able to provide a tighter integration between Constructor and the IDE in order to facilitate building an application without having to continually switch back and forth between the two environments. One of our goals in this direction is to allow the user to build the UI for their application, and get it up and running without having too write any code, so that they can quickly get something working.

Dave: WebBurst allows you to add graphic prettiness and animation to your applets in a way not currently supported by Constructor. Will Constructor be moving in this direction?

Robin: Many of these capabilities can be delivered by providing a richer set of objects that can be manipulated within Constructor. It is certainly one of our goals to provide a much richer set of objects that can be utilized in the construction of an application's UI. Up until now we have been focusing on getting the core set of functionality in place. Once that task has been accomplished, we can start to provide richer capabilities that are not in the current versions of Constructor or PowerPlant.

We also are looking into mechanisms that will allow the user to introduce their custom classes into Constructor so that they have the ability to extend the set of capabilities that are available to them. CPPb's currently allow you to do this but you cannot see how your classes would render, so the goal is provide a mechanism that would allow the drawing code from your classes to be available when your classes are included in a layout, thus allowing the visual manipulation to take on a more realistic representation than what is currently available in Constructor.

This mechanism could also potentially allow third party developers to create class libraries that could be used from within Constructor.

Dave: Does this mean that I'll be able to import my own code into Constructor?

Robin: This has always been one of the thorny issues with Constructor, and any interface builder written with C++, for that matter.

Up until now we have had the CPPb mechanism which at least allows the user to setup the values they care about for their custom classes and to manipulate their layout, but the manipulation process has been a little unsatisfactory because their classes have only been represented in a layout as a rectangular box, which doesn't exactly give you a feel for the appearance of the interface under construction.

The problem has been that with C++ there really is no clean way to get the users code included into Constructor, particularly the rendering aspects of the code, which is what you care about during the layout process.

In order to address this, Eric is working on a mechanism that would allow the users custom classes to handle the rendering of these classes within a layout, thereby making the layout process more visual than is currently available. We hope to make this available in a future version of Constructor in order to improve the layout process for custom classes.

As part of this effort we would also like to improve the CPPb mechanism so that it has greater flexibility, particularly in terms of the types that are available for constructing CPPbs.

Dave: What kind of changes do you see in future versions of Constructor?

Robin: Well as we've already mentioned, you can expect to see Constructor move more towards the application building realm, by allowing the user to perform many of the operations required to build an application directly from within Constructor. It is also our intent to establish a tighter integration with the IDE so that tasks, such as running or building the application, can be initiated from within Constructor.

We will also continue to develop the feature set of Constructor in order to deliver more power and flexibility to the application development process.

For example, we are planning on enhancing Constructor's abilities to manage the various assets that come into play in the application building process. This basically can be viewed as a cataloging process that will provide the user with a centralized location for storing, browsing, and retrieving the various elements that are used to build both the UI for the application, as well as the application itself. This cataloging mechanism would be flexible enough to allow the user to drag elements from a catalog into a layout, or conversely to be able to drag a collection of elements from a layout back into the catalog for future reuse. The contents of the catalog would not be restricted to simple widgets, but would include all of the pieces needed in the construction process.

Another area we intend to improve is the inspection and editing of object properties, some of which you can get an early glimpse of by looking at the Java version of Constructor. This mechanism

will be used in the PowerPlant version as well, and will be enhanced to provide a richer set of property editing capabilities.

As shown in the latest version of Constructor, with the inclusion of the icon editor, we want to continue delivering more of the resource editing capabilities that are needed in order to successfully construct an application.

We are definitely not standing still on Constructor. It is our goal to continue moving the tool forward in an effort to make it more powerful, flexible, and extensible. We also want to make it more accessible to novice users by continuing to improve its interface and the set of capabilities it delivers.

Clint: Since we are using live Java objects for displaying the interface, we plan to allow installation of user classes in Constructor, so that one could build interfaces with live instances of their classes. Along with this, we would like to provide a mechanism for added data in Constructor which could be streamed into user classes at runtime, similar to custom types in the PowerPlant Constructor.

Eric: The major directions for Constructor in the future are to take more of the gruntwork out of PowerPlant programming and to extend the resource-editing capabilities of the tool.

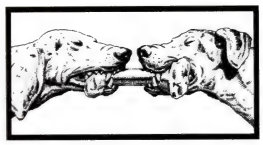
In the end, this is a tool to help users write great applications. Many of the features that you've seen implemented or planned are the direct result of user suggestions or comments. I invite people to contact me directly by e-mail (scouten@metrowerks.com) if they have new ideas that they'd like to see in future Constructors.

Eric Scouten is the technical lead for Constructor at Metrowerks. He lives in St Paul, Minnesota, and works... well, almost anywhere. Portions of Constructor have been written in planes, trains, and automobiles all over the country. When not engaged in a late-night coding frenzy, Eric is likely to be found behind the lens of his camera, shooting slides of landscapes, barns or little furry animals. (Despite a concerted effort, he hasn't yet captured a picture of the elusive dogcow.)

Clint Popetz went from calculating Clebsch-Gordon coefficients as a student of physics to battling MacTCP while working on NCSA Telnet for the Mac. Now he spends his time writing code with one hand and keeping the mouse away from his 8 month old daughter with the other.

Robin Mair worked at Apple for 7 years, primarily in the Developer Tools group, the last few years were spent working on an interface builder for Dylan which, sadly, was never completed. His primary interests are in interface builders and user interfaces, but he also likes to surf, especially in Hawaii.

PROGRAMMER'S CHALLENGE



By Bob Boonstra

Router Rules

This month's Challenge is based on a suggestion by Peter Lewis and is motivated by a real-world problem. A certain university has a B-class IP subnet, let's call it 199.232.*.* (with apologies to the real-world owner of that subnet). The subnet is broken down into 256 networks for the various faculties and departments, each one having 256 IP numbers. So, for example, the computer club might have 199.232.101.*. Our hypothetical university is charged for communications based on volume, so some of these networks are allowed to talk to the outside world, and others are not. Outside access is controlled by programming a router with a sequence of rules, each of which allows or denies access to some subset of IP numbers. A rule consists of a (mask, value, allow) triplet. For example, say the networks (in hex) 01, 03, 41, 43 are allowed out, and all the rest are barred. The rules could be simply

```
FF, 01, allow
FF, 03, allow
FF, 41, allow
FF, 43, allow
00, 00, deny
```

But this could be simplified to

```
BD, 01, allow
00, 00, deny
```

Your objective for this Challenge is to quickly generate a small sequence of rules that allows outside network access to only a specified set of networks. The prototype for the code you should write is

```
enum {kDeny=0, kAllow=1};

typedef struct Rule {
  long mask;
  long value;
  long allow;/* 0 == deny, 1== allow */
} Rule;

long RouterRules(
  long allowedValues[],
  long numAllowedValues,
  long numBits,
  Rule rulesArray[],
  long maxRules
);
```

The array `allowedValues` is the set of `numAllowedValues` networks that are to be given outside network access. All other networks should be denied access. Instead of being limited to 8 bits as in the example above, network values have **numBits** bits. Your code should generate a sequence of rules that provides access to these networks, and no others. The rule sequence should be as short as possible and stored in **rulesArray**, which is allocated by the caller and is of size **maxRules**. Your code should return the number of rules generated, or return -1 if it cannot find a solution no longer than **maxRules**.

Rules will be triggered by the router in the order provided by your solution, and the first rule to fire for a given network will apply. At least one rule must fire for any possible network value. For example, if **numBits==3**, and we want to allow access to networks 0, 2, 3, 6, and 7, you could use the following rules:

```
3, 1, deny
6, 4, deny
7, 7, allow
```

To encourage code that generates both fast and short solutions, the ranking will be based on minimizing the following function of execution time on my 8500/150 and the number of rules generated:

```
score = (number of rules generated) + (execution time in
seconds) / 2
```

This will be a native PowerPC Challenge, using the latest CodeWarrior environment. Solutions may be coded in C, C++, or Pascal.

Two Months Ago Winner

Congratulations to **Xan Gregg** (Durham, N.C.), for submitting the fastest entry to the ByteCode Interpreter Programmer's Challenge, narrowly beating out the second-place entry by **Ernst Munter**. The Challenge was to write an interpreter for a subset of the byte code language implemented by the Java Virtual Machine. The Challenge rules pointed to the Java Virtual Machine Specification for a description of the opcodes, with some exclusions about the opcodes and features that were to be implemented, and with the significant simplifying assumption that the Virtual Machine need only deal with a single class file. Of the five solutions submitted, three worked correctly for all test cases, one worked for all but one test case, and the fifth was acknowledged by the author to be incomplete.

The rules for September permitted the use of assembly language, and Xan was the only contestant to submit a solution that took advantage of this. After parsing the header to identify the constants, fields, and methods contained in the class file, the

solution dispatches and executes each opcode. As described by the comments in the code, the main execution loop contains a table with 32 bytes of PowerPC instructions implementing each opcode. Opcodes that require more code than will fit into the table entry overflow to code outside the table. Particular features that you might want to examine in the code include the implementation of the jump table and the pseudo-opcode ExitCode used to trigger a return to the calling routine. Congratulations to Xan on an elegant, efficient, and instructive solution. Several readers commented that they learned quite a bit about Java from implementing a Virtual Machine. Congratulations as well to everyone who participated in this more difficult than usual Challenge!

The table below summarizes the results for each entry, including execution time in milliseconds, code size, and data size. An asterisk indicates a test case that was not successfully completed by a solution. Numbers in parenthesis after a person's name indicate that person's cumulative point total for all previous Challenges, not including this one.

Name	Language	Test 1	Test 2	Test 3	Test 4	Time	Code	Data
Xan Gregg (92)	C/Assembly	31088	12923	24607	45190	113808	10064	171
Ernst Munter (214)	C++	18661	14664	28260	52496	114081	6260	879
Turlough O'Connor	C++	23073	15946	30503	57444	126967	14536	5818
Conor MacNeill	C++	44446	*	43298	84020	*	82536	10909

The test code was contained in standard Java applets, which allowed me to use the interpreters supplied with the Symantec and Metrowerks environments to confirm expected results. This also allowed a comparison of the execution time of the mini Virtual Machines submitted as solutions with the execution time of the commercial interpreters. Results for the same four test cases used to score the solutions are presented below for the Applet Viewer provided with Symantec Cafe (version 1.0) and for the Metrowerks Java interpreter provided with CodeWarrior 9. While the comparison is not entirely fair because of the simplifying assumptions used in this Challenge, the table indicates that three of the four solutions were faster than both of these interpreters. Although CodeWarrior 10 has not been finalized as this column is being written, it should be available at publication, and Metrowerks was kind enough to give me a preview of the Java interpreters available in that release. For my limited set of test cases, the CW10 version of Metrowerks Java was approximately 25% faster than the CW9 version. Even more impressive was the Just-In-Time version of the interpreter, which (I assume) first compiles the byte-coded instructions into PowerPC instructions before execution. My tests suggest that Metrowerks Java JIT executes an order of magnitude faster than the CW9 interpreter (not counting the preprocessing compilation time, which my tests did not measure).

Product	Test 1	Test 2	Test 3	Test 4	Time
Symantec Cafe 1.0	41000	34000	62000	134000	271000
Metrowerks Java (CW9)	24175	18714	35075	68077	146041
Metrowerks Java (CW10 preview)	18336	14253	27494	52285	112368
Metrowerks Java JIT (CW10 preview)	5487	1549	2560	5977	15573

TOP 20 CONTESTANTS

Here are the Top 20 Contestants for the Programmer's Challenge. The numbers below include points awarded over the 24 most recent contests, including points earned by this month's entrants.

Rank	Name	Points
1.	Munter, Ernst	193
2.	Gregg, Xan	112
3.	Larsson, Gustav	87
4.	Lengyel, Eric	40
5.	[Name deleted]	40
6.	Lewis, Peter	32
7.	Boring, Randy	27
8.	Beith, Gary	24
9.	Kasparian, Raffi	22
10.	Vineyard, Jeremy	22
11.	Cutts, Kevin	21
12.	Picao, Miguel Cruz	21
13.	Brown, Jorg	20
14.	Gundrum, Eric	20
15.	Karsh, Bill	19
16.	Stenger, Allen	19
17.	Cooper, Greg	17
18.	Mallett, Jeff	17
19.	Nevard, John	17
20.	Nicolle, Ludovic	14

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place .....20 points
2nd place .....10 points
3rd place .....7 points
4th place.....4 points
5th place.....2 points
finding bug.....2 points
suggesting Challenge.....2 points

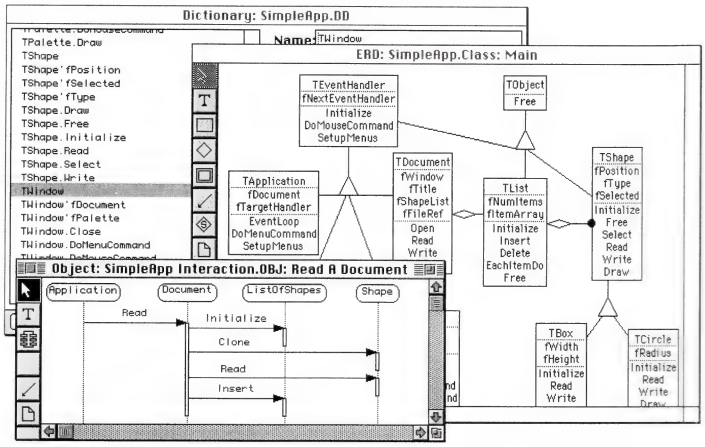
Here is Xan's winning solution:

JAVAMINIVM.C

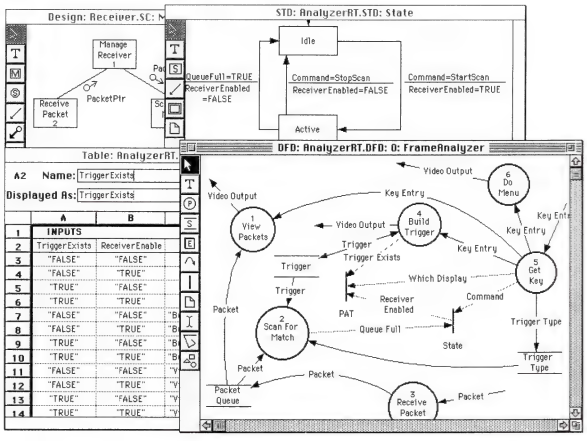
COPYRIGHT 1996, XAN GREGG

```
/*
The core of the interpreter is written in mostly PowerPC assember, and the parsing of
the tables is done in C before interpreting begins.  I create tables of relevant data for
constants, fields, and methods.  The constants data is just the address of the Java
constant_pool data for that index.  For fields and methods I allocate structs with the
useful data and insert a pointer to the struct into the field's or method's constant_pool
entry.

So, for some up-front overhead and memory usage (16 bytes per field or method), the
interpreter has less work to do when dealing with a field or method.

String objects are pointers to the constant_pool payload, just as the provided test code
expects.

Objects in general are greatly simplified because of the one class limitation.  No type
information is stored with any object in the heap.

Of the 10M of provided heapSpace, I use 512K for the Java heap, 512K for the Java
stacks (data and return), and the rest is available for the constant, field, and method
tables, which will consume less that 20 * NumConstants bytes.

The heap size and stack size can be set by the macros HEAP_K and STACK_K below.
No garbage collection is performed on the heap, so if there is lots of object creation, a
larger heap may be needed.

The VM has two stacks, a data stack and a return stack.  The data stack grows up from
the start of the stack space, and the return stack grows down from the end of the stack
space.  The data stack is used for parameters and normal Java stack operations.  The
return stack is used method nesting.  Each call makes a three-entry stack frame
consisting of the return address, frame pointer (start of locals), and tsBase (the method
base for use by tableswitch and lookupswitch).

Having separate stacks made the parameter passing easier, but now I realize that with
some extra work, I could have just put the return frame into added local variables.

The Top-Of-Stack is kept in a register.  This requires a little handshaking when the
interpreter calls another function, which doesn't have access to the TOS register.

The core of the interpreter is in the assembly routine, StartVM.  Its main loop fetches
and dispatches opcodes, as expected.  It includes a table with a 32-byte entry for each
opcode (up through 209).  An entry consists of code to implement the opcode.  Most
of them fit in 32-bytes, and any extra space is used to prime the dispatch loop.

Opcodes that take more than 7 instructions to implement can spill over to code outside
of the table.  Opcodes that are particularly complex (like multianewarray) are
implemented with C functions.

Limitations:
I try to safely ignore operations involving long, double, and float operands by treating
these types as taking 0 bytes each.  Most instructions become a NOP, and others are
very simple (i2d become a POP).  However, I have realized one flaw to this system:
The untyped instructions (like DUP) may try to operate on the unsupported types.

It's too late to fix now, but since the challenge will try to avoid unsupported types, it's
unlikely the stack manipulation of them will be needed.

Future:
With the PowerPC code done for each opcode, it would not be too tough to make a
compiler that would string together the PowerPC code for each opcode in each
method.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Java types
typedef signed char s1;
typedef signed short s2;
typedef signed long s4;
typedef unsigned char u1;
typedef unsigned short u2;
typedef unsigned long u4;

// Set these if necessary
#define HEAP_K  512
#define STACK_K 512

// the CONSTANT_ id's
enum {C_Utf8 = 1, C_Unicode, C_Integer, C_Float, C_Long,
     C_Double, C_Class, C_String, C_Fieldref,
     C_Methodref, C_InterfaceMethodref, C_NameAndType};

const u2 ACC_STATIC = 0x0008;

// The contents of my tables
typedef u1 *ConstantData;

typedef struct
{
  u2   *fieldInfoP;
  long offset;        // in bytes
  long size;          // in bytes
  u1   *type;         // ptr to sig constant
} FieldData;

typedef struct
{
  u2   *methodInfoP;
  u2   *codeAttrP;
  u1   *codeP;
  u4   paramCount;   // in bytes
} MethodData;

// prototypes
void JavaMiniVM(void *constant_pool, void *fields,
  void *methods, void *classFile,
  long methodToExecute, void *heapSpace,void *returnStack);

static short IndexConstants(void *constants,
                ConstantData *indexArray);
static short IndexFields(void *fields);
static short IndexMethods(void *methods);
static void CreateAndPush(void);
static void AllocateStaticFields(void);
static void ResolveFields(void);
static void ResolveMethods(void);
static void CountParams(MethodData *methodP);
asm static long StartVM(register MethodData *methodP);
static void PushConstant(long n);
static u1 *TableSwitch(u1 *ip, long tsBase, long n);
static u1 *LookupSwitch(u1 *ip, long tsBase, long key);
static long GetNewArray(long type, long size);
static long GetANewArray(long size);
static void PushMultiANewArray(long classIndex,
              long numDimensions);

// globals
static long     NumMethods;
static long     NumFields;
static long     NumConstants;
static MethodData *Methods;
static FieldData  *Fields;
static ConstantData *Constants;
static long     *FP; // frame pointer
static long     *SP; // stack pointer
static long     *S0; // stack base
static long     *RP; // return stack pointer
static long     *R0; // return stack base
static long     *HP; // heap pointer
static long     *H0; // heap base
static long     TotalStatic;       // in bytes
static long     TotalNonStatic;    // in bytes
static long     LastField;
static long     LastMethod;

/* java types
typedef struct
{
0  u2 attribute_name;
2  u4 attribute_length;
6  u1 info[1];
} attribute_info;
```

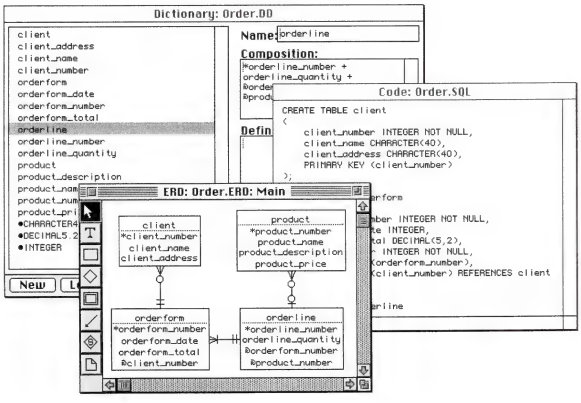
Shift your development projects into high gear with MacA&D software engineering tools!



OOA/OOD includes OMT, Booch, Coad/Yourdon, Shlaer/Mellor...



Structured A&D using Yourdon/DeMarco, Gane/Sarson, Hatley/Pirbhai...



Data modeling and SQL generation for Information Engineering, Chen...

MacA&D

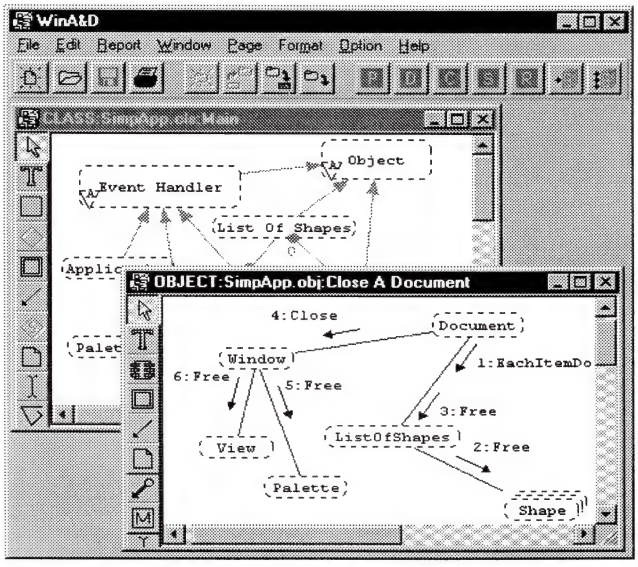
Successful software development requires an understanding of what needs to be done, a plan for how to do it and a structured approach to completing the job. MacA&D can help by simplifying system analysis and requirements specification, automating popular modeling techniques for designing your software and generating source code from your design to give you a head start on implementation. Throughout the process, extensive verification reports will catch errors early and help you produce a quality product.

Powerful capabilities are just one MacA&D advantage. It's easy-to-use and includes an extensive set of examples and tutorials. MacA&D and WinA&D products can share documents, so your project team can use Macintosh, Solaris, HP-UX, Windows 95 or Windows NT computers. You can even switch design methods or pick and mix the best of each, so your design work always stays current.

Software Engineering

Structured Analysis & Design
Object-Oriented Analysis & Design
Real-Time State & Task Modeling
Data & Screen Modeling
Integrated Code Editing & Browsing
Multi-User Dictionary & Requirements
Code to Design & Design to Code

MacAnalyst $ 995	MacA&D $1995
MacDesigner $ 995	Translator $ 495



New WinA&D for Windows 95 & Windows NT!

Call for MacA&D or WinA&D brochures or fax 515-752-2435 casetools@aol.com.

Excel Software
515-752-5359

```
typedef struct
{
0  u2 access_flags;
2  u2 name_index;
4  u2 signature_index;
6  u2 attribute_count;
8  attribute_info   attributes[1];
} field_info;

typedef struct
{
0  u2 access_flags;
2  u2 name_index;
4  u2 signature_index;
6  u2 attribute_count;
8  attribute_info   attributes[1];
} method_info;

typedef struct
{
0  u1 tag;
1  u2 length;
3  u1 bytes[1];
} CONSTANT_Utf8_info
*/

#define PUSH(a)   *SP++ = (a)

#define MethodIsStatic(n)  \
   ((*Methods[n].methodInfoP & ACC_STATIC) != 0)
```

JavaMiniVM

```
void JavaMiniVM(void *constant_pool, void *fields,
   void *methods, void *classFile,
   long methodToExecute, void *heapSpace,void *returnStack)
{
   //allocate java heap
   H0 = (long *) heapSpace;
   HP = H0;
   // allocate java stack (stack grows up in memory)
   S0 = H0 + HEAP_K * 256L;    // 256 longs = 1K
   SP = S0;

   // return stack based at end of stack space (grows down)
   R0 = S0 + STACK_K * 256L;   // 256 longs = 1K
   RP = R0;

   Constants = (ConstantData *) R0;
   NumConstants = IndexConstants(constant_pool, Constants);

   Fields = (FieldData *) (Constants + NumConstants);
   NumFields = IndexFields(fields);
   Methods = (MethodData *) (Fields + NumFields);
   NumMethods = IndexMethods(methods);

   ResolveFields();
   ResolveMethods();
   AllocateStaticFields();
   if (!MethodIsStatic(methodToExecute))
     CreateAndPush();  // so it needs an obj
   *(long *) returnStack =
        StartVM(&Methods[methodToExecute]);

}
```

IndexConstants

```
// create on index into the constant_pool
static short IndexConstants(void *constants,
                  ConstantData *indexArray)
{
   short  n, i;
   u1   *p;

   // list is preceded by length
   n = * ((u2 *) constants - 1);
   p = (u1 *) constants;
   LastField = 0;
   LastMethod = 0;
   for (i = 1; i < n; i++)
```

```
    {
      indexArray[i] = p;
      switch (*p)
      {
        case C_Utf8:  p += 3 + *(u2*)(p+1); break;
        case C_Unicode: p += 3 + *(u2*)(p+1); break;
        case C_Integer: p += 5; break;
        case C_Float: p += 5; break;
        case C_Long:  p += 9; i += 1; break;
        case C_Double:  p += 9; i += 1; break;
        case C_Class: p += 3; break;
        case C_String:  p += 3; break;
        case C_Fieldref: p += 5; LastField = i; break;
        case C_Methodref: p += 5; LastMethod = i; break;
        case C_InterfaceMethodref: p += 5; break;
        case C_NameAndType: p += 5; break;
      }
    }
    indexArray[0] = p;   //useful for getting to 'this_class'
    return n;
}
```

IndexFields

```
// Gather FieldData about each field
static short IndexFields(void *fields)
{
  short  n, i;
  u1   *p;
  u1   *sigP;
  u2   sigIndex;
  FieldData  *fieldP;
  u1   c;
  long staticOffset = 0;
  long memberOffset = 0;

  // list is preceded by length
  n = * ((u2 *) fields - 1);
  p = (u1 *) fields;
  fieldP = Fields;
  for (i = 0; i < n; i++)
  {
    short  count; // attribute count

    fieldP->fieldInfoP = (u2 *) p;
    sigIndex = *(u2 *) (p + 4);
    sigP = Constants[sigIndex];
    c = *(sigP + 3);
    if (c == 'L' || c == 'D' || c == 'F')
      fieldP->size = 0;
    else
      fieldP->size = 4;
    fieldP->type = sigP;
    if ((*fieldP->fieldInfoP & ACC_STATIC) != 0)
    {
      fieldP->offset = staticOffset;
      staticOffset += fieldP->size;
    }
    else
    {
      fieldP->offset = memberOffset;
      memberOffset += fieldP->size;
    }



    // skip attributes
    count = * (u2 *) (p + 6);
    p += 8;
    while (count > 0)
    {
      count -= 1;
      p += 6 + *(u4 *) (p+2);
    }
    fieldP += 1;
  }
  TotalStatic = staticOffset;
  TotalNonStatic = memberOffset;
  return n;
}
```

IndexMethods

```
// Gather MethodData about each method
static short IndexMethods(void *methods)
{
  short  n, i;
  u1   *p;
  MethodData *methodP;

  // list is preceded by length
  n = * ((u2 *) methods - 1);
  p = (u1 *) methods;
  methodP = Methods;
  for (i = 0; i < n; i++)
  {
    short  count; // attribute count
    u2   nameIndex;
    u1   *nameEntryP;

    methodP->methodInfoP = (u2 *) p;
    CountParams(methodP);
    count = * (u2 *) (p + 6);
    p += 8;
    while (count > 0)
    {
      nameIndex = * (u2 *) (p);
      nameEntryP = Constants[nameIndex];
      if (*(u2*)(nameEntryP + 1) == 4
          && *(long *) (nameEntryP + 3) == 'Code')
      { // this is the code attr
        methodP->codeAttrP = (u2 *) p;
        methodP->codeP = (u1 *) (p + 14);
      }
      count -= 1;
      p += 6 + *(u4 *) (p+2);
    }
    methodP += 1;
  }
  return n;
}
```

CreateAndPush

```
// allocate 'this' and push a reference to it
static void CreateAndPush(void)
{
  *SP = (long) HP;
  SP ++;
  // heap already initialized to zeros
  HP = (long *) ((long) HP + TotalNonStatic);
}
```

AllocateStaticFields

```
static void AllocateStaticFields(void)
{
// heap already initialized to zeros
  HP = (long *) ((long) HP + TotalStatic);
}
```

ResolveFields

```
// Change the payload of CONSTANT_Fieldref items to be a
// pointer into the FieldData array
static void ResolveFields(void)
{
  short  i, j;
  u1   *p;
  u1   *q;
  ConstantData  *constantP;
  FieldData  *fieldP;
  u2   nameTypeIndex;
  u2   nameIndex;

  constantP = Constants+1;
  for (i = 1; i <= LastField; i++)
  {
    p = *constantP++;
    if (*p == C_Fieldref)
    {
      fieldP = Fields;
      nameTypeIndex = *(u2*) (p + 3);
```

```
        q = Constants[nameTypeIndex];
        nameIndex = *(u2*) (q + 1);
        for (j = 0; j < NumFields; j++)
        {
          if (*(fieldP->fieldInfoP+1) == nameIndex)
          { // found matching field ref, change it
            *(u4*) (p+1) = (u4) fieldP;
            break;
          }
          fieldP += 1;
        }
      }
    }
}
```

ResolveMethods

```
// Change the payload of CONSTANT_Methodref items to be a
// pointer into the MethodData array
static void ResolveMethods(void)
{
  short  i, j;
  u1   *p;
  u1   *q;
  ConstantData  *constantP;
  MethodData *methodP;
  u2   nameTypeIndex;
  u2   nameIndex;
  u2   sigIndex;

  constantP = Constants+1;
  for (i = 1; i <= LastMethod; i++)
  {
    p = *constantP++;
    if (*p == C_Methodref)
    {
      methodP = Methods;
      nameTypeIndex = *(u2*) (p + 3);
      q = Constants[nameTypeIndex];
      nameIndex = *(u2*) (q + 1);
      sigIndex = *(u2*) (q + 3);
      for (j = 0; j < NumMethods; j++)
      {
        if (methodP->methodInfoP[1] == nameIndex
          && methodP->methodInfoP[2] == sigIndex)
        { // found matching method ref, change it
          *(u4*) (p+1) = (u4) methodP;
          break;
        }
        methodP += 1;
      }
    }
  }
}
```

CountParams

```
static void CountParams(MethodData *methodP)
{
  long sigIndex;
  long paramCount;
  u1   *sigPtr;
  u2   sigLength;
  u1   ch;

  sigIndex = *(methodP->methodInfoP + 2);
  sigPtr = Constants[sigIndex];
  sigLength = * (u2 *) (sigPtr + 1);
  sigPtr += 4;        // skip tag, length and '('
  if ((*methodP->methodInfoP & ACC_STATIC) == 0)
    paramCount = 4; // implicit class object parameter
  else
    paramCount = 0;
  while (1)
  {
    ch = *sigPtr;
    if (ch == ')')
      break;
    paramCount += 4;
    if (ch == 'D' || ch == 'J' || ch == 'F')
      paramCount -= 4;  // these are 0-byte types
    else if (ch == 'L')
    { // skip class name


      while (*++sigPtr != ';')
        ;
    }
    else if (ch == '[')
    { // we don't care what it's an array of
      while (*++sigPtr == '[')
        ;
      if (*sigPtr == 'L')
        while (*++sigPtr != ';')
          ;
    }
    sigPtr += 1;
  }
  ch = *++sigPtr;
  methodP->paramCount = paramCount;
}

#define slwi(r, n)          \
  rlwinm r, r, n, 0, 31-n;

#define times4(r)         \
  rlwinm r, r, 2, 0, 29;

// the number after 'used' indicates the number of
// instructions used so far in this slot.

#define used0           \
  lbz   opcode, 0(ip);    \
  slwi(opcode, 5)       \
  add   a, base, opcode;  \
  mtctr  a;           \
  addi ip, ip, 1;       \
  bctr;             \
  nop; nop;

#define used1           \
  lbz   opcode, 0(ip);    \
  slwi(opcode, 5)       \
  add   a, base, opcode;  \
  mtctr  a;           \
  addi ip, ip, 1;       \
  bctr;             \
  nop;

#define used2           \
  lbz   opcode, 0(ip);    \
  slwi(opcode, 5)       \
  add   a, base, opcode;  \
  mtctr  a;           \
  addi ip, ip, 1;       \
  bctr;

#define used3           \
  lbz   opcode, 0(ip);    \
  slwi(opcode, 5)       \
  add   a, base, opcode;  \
  mtctr  a;           \
  b   next4;

#define used4           \
  lbz   opcode, 0(ip);    \
  slwi(opcode, 5) \
  add   a, base, opcode;  \
  b   next3;

#define used5           \
  lbz   opcode, 0(ip);    \
  slwi(opcode, 5)       \
  b   next2;

#define used6           \
  lbz   opcode, 0(ip);    \
  b   next1;

#define used7           \
  b   next0;

#define pushtos     \
  stw   tos, 0(sp);\
  addi sp, sp, 4;

#define poptos      \
  lwzu tos, -4(sp)

#define pushi(n)      \
```

```
  stw    tos, 0(sp);\
  li   tos, n;     \
  addi sp, sp, 4;

#define pushr(r)     \
  stw    tos, 0(sp);\
  mr   tos, r;     \
  addi sp, sp, 4;

// The initial return address
// points here.
ulExitCode[1] =
{203};
```

StartVM

```
asm static long
StartVM(register
MethodData *methodP)
{
  register long *sp;
  register long *rp;
  register long *fp;
  register long *h0;
  register long *cp;
  // Constants
  register ul   *ip;
  // instruction ptr
  register ul
opcode;
  register long
tsBase;
// for tableswitch  padding
  register long a;
  register long b;
  register long c;
  register long base;
// base of our opcode table
  register long tos;

  fralloc
  lwz    sp, SP
  lwz    rp, RP
  lwz    h0, H0
  lwz    cp,
Constants
  lwz    a, ExitCode
  stwu a, -4(rp)
  // push initial return ip
  lwzu tos, -4(sp)
// put TOS into a reg.

// get address of opcode table
  bl   nowhere
nowhere:
  mflr base
// offset hard-coded since
// MW won't subtract labels
  addi base, base,
22*4 //table - nowhere

start_method:
  stwu fp, -4(rp)

  pushtos
// put tos (last param) in
// memory
  mr   fp, sp
// FP points to first parameter
  lwz    a, methodP-
>paramCount
  subf fp, a, fp
// allocate space for locals
  lwz    b, methodP-
>codeAttrP
  lhz    c, 8(b)
// max locals
  times4(c)
  subf c, a, c
  add    sp, sp, c
// init ip
  lwz    ip, methodP-
>codeP
// init tsBase
  andi.  tsBase, ip,
3
  stwu tsBase, -4(rp)
next0:
  lbz    opcode,
0(ip)
next1:
  slwi(opcode, 5)
next2:
  add    a, base,
opcode
next3:
  mtctr  a
next4:
  addi ip, ip, 1
  bctr

table:
  // 0 nop
  used0
  // 1 aconst_null
  pushi(0)
  used3
  // 2 iconst_m1
  pushi(-1)
  used3
  // 3 iconst_0
  pushi(0)
  used3
  // 4 iconst_1
  pushi(1)
  used3
  // 5 iconst_2
  pushi(2)
  used3
  // 6 iconst_3
  pushi(3)
  used3
  // 7 iconst_4
  pushi(4)
  used3
  // 8 iconst_5
  pushi(5)
  used3
  // 9 lconst_0
  used0
  // 10 lconst_1
  used0
  // 11 fconst_0
  used0
  // 12 fconst_1
  used0
  // 13 fconst_2
  used0
  // 14 dconst_0
  used0
  // 15 dconst_1
  used0
  // 16 bipush
  lbz    a, 0(ip)
  extsb  a, a
  addi ip, ip, 1
  pushr(a)
  used6
  // 17 sipush
  lha    a, 0(ip)
  addi ip, ip, 2
  pushr(a)
  used5
  // 18 ldc1
  lbz    r3, 0(ip)
  addi ip, ip, 1
  pushtos
  b ldc_more
  nop;nop;nop
  // 19 ldc2
  lhz    r3, 0(ip)
  addi ip, ip, 2
  pushtos
  b ldc_more
  nop;nop;nop
  // 20 ldc2w
  lhz    r3, 0(ip)
  addi ip, ip, 2
  pushtos
  b ldc_more
  nop;nop;nop
  // 21 iload
  lbz    a, 0(ip)
  addi ip, ip, 1
  times4(a)
  // convert index to offset
  pushtos
  lwzx tos, a, fp
  used6
  // 22 lload
  used0
  // 23 fload
  used0
  // 24 dload
  used0
  // 25 aload
  lbz    a, 0(ip)
  addi ip, ip, 1
  times4(a)
  // convert index to offset
  pushtos
  lwzx tos, a, fp
  used6
  // 26 iload_0
  pushtos
  lwz    tos, 0(fp)
  used3
  // 27 iload_1
  pushtos
  lwz    tos, 4(fp)
  used3
  // 28 iload_2
  pushtos
  lwz    tos, 8(fp)
  used3
  // 29 iload_3
  pushtos
  lwz    tos, 12(fp)
  used3
  // 30 lload_0
  used0
  // 31 lload_1
  used0
  // 32 lload_2
  used0
  // 33 lload_3
  used0
  // 34 fload_0
  used0
  // 35 fload_1
  used0
  // 36 fload_2
  used0
  // 37 fload_3
  used0
  // 38 dlload_0
  used0
  // 39 dload_1
  used0
  // 40 dload_2
  used0
  // 41 dload_3
  used0
  // 42 aload_0
  pushtos
  lwz    tos, 0(fp)
  used3
  // 43 aload_1
  pushtos
  lwz    tos, 4(fp)
  used3
  // 44 aload_2
  pushtos
  lwz    tos, 8(fp)
  used3
  // 45 aload_3
  pushtos
  lwz    tos, 12(fp)
  used3
  // 46 iaload
  lwzu b, -4(sp)
  // get array address
  mr   a, tos
  addi b, b, 4
  times4(a)
  // convert index to offset
  lwzx tos, a, b
  used5
  // 47 laload
  lwzu tos, -8(sp)
  // drop2
  used1
  // 48 faload
  lwzu tos, -8(sp)
  // drop2
  used1
  // 49 daload
  lwzu tos, -8(sp)
  // drop2
  used1
  // 50 aaload
  lwzu b, -4(sp)
  // get array address
  mr   a, tos
  addi b, b, 4
  times4(a)
  // convert index to offset
  lwzx tos, a, b
  used5
  // 51 baload
  lwzu b, -4(sp)
  // get array address
  addi b, b, 4
  lbzx tos, tos, b
  extsb  tos, tos
  used4
  // 52 caload
  lwzu b, -4(sp)
  // get array address
  mr   a, tos
  addi b, b, 4
  slwi(a, 1)
  lhzx tos, a, b
  used5
  // 53 saload
  lwzu b, -4(sp)
  // get array address
  mr   a, tos
  addi b, b, 4
  slwi(a, 1)
  // convert index to offset
  lhax tos, a, b
  used5
  // 54 istore
  lbz    a, 0(ip)
  addi ip, ip, 1
  times4(a)
  // convert index to offset
  stwx tos, a, fp
  lwzu tos, -4(sp)
  used5
  // 55 lstore
  used0
  // 56 fstore
  used0
  // 57 dstore
  used0
  // 58 astore
  lbz    a, 0(ip)
  addi ip, ip, 1
  times4(a)
  // convert index to offset
  stwx tos, a, fp
  lwzu tos, -4(sp)
  used5
  // 59 istore_0
  stw    tos, 0(fp)
  lwzu tos, -4(sp)
  used2
  // 60 istore_1
  stw    tos, 4(fp)
  lwzu tos, -4(sp)
  used2
  // 61 istore_2
  stw    tos, 8(fp)
  lwzu tos, -4(sp)
  used2
  // 62 istore_3
  stw    tos, 12(fp)
  lwzu tos, -4(sp)
  used2
  // 63 lstore_0
  used0
  // 64 lstore_1
  used0
  // 65 lstore_2
  used0
  // 66 lstore_3
  used0
  // 67 fstore_0
  used0
  // 68 fstore_1
  used0
  // 69 fstore_2
  used0
  // 70 fstore_3
  used0
  // 71 dstore_0
  used0
  // 72 dstore_1
  used0
  // 73 dstore_2
  used0
  // 74 dstore_3
  used0
  // 75 astore_0
  stw    tos, 0(fp)
  lwzu tos, -4(sp)
  used2
  // 76 astore_1
  stw    tos, 4(fp)
  lwzu tos, -4(sp)
  used2
  // 77 astore_2
  stw    tos, 8(fp)
  lwzu tos, -4(sp)
  used2
  // 78 astore_3
  stw    tos,12(fp)
  lwzu tos, -4(sp)
  used2
  // 79 iastore
  lwzu b, -4(sp)
  // index
  lwzu c, -4(sp)
  // array address
  times4(b)
  // index to offset
  addi c, c, 4
  // skip arraylength field
  stwx tos, b, c
  lwzu tos, -4(sp)
  used6
  // 80 lastore
  lwzu tos, -8(sp)
  // drop2
  used1
  // 81 fastore
  lwzu tos, -8(sp)
  // drop2
  used1
  // 82 dastore
  lwzu tos, -8(sp)
  // drop2
  used1
  // 83 aastore
  lwzu b, -4(sp)
  // index
  lwzu c, -4(sp)
  // array address
  times4(b)
  // index to offset
  addi c, c, 4
  // skip arraylength field
  stwx tos, b, c
  lwzu tos, -4(sp)
  used6
  // 84 bastore
  lwzu b, -4(sp)
  // index
  lwzu c, -4(sp)
```

```
// array address
addi c, c, 4
// skip arraylength field
stbx tos, b, c
lwzu tos, -4(sp)
used5
// 85 castore
lwzu b, -4(sp)
// index
lwzu c, -4(sp)
// array address
slwi(b, 1)
// index to offset
addi c, c, 4
// skip arraylength field
sthx tos, b, c
lwzu tos, -4(sp)
used6
// 86 sastore
lwzu b, -4(sp)
// index
lwzu c, -4(sp)
// array address
slwi(b, 1)
// index to offset
addi c, c, 4
// skip arraylength field
sthx tos, b, c
lwzu tos, -4(sp)
used6
// 87 pop
lwzu tos, -4(sp)
used1
// 88 pop2
lwzu tos, -8(sp)
used1
// 89 dup
pushtos
used2
// 90 dup_x1
lwz    a, -4(sp)
stw    tos, -4(sp)
stw    a, 0(sp)
addi sp, sp, 4
used4
// 91 dup_x2
lwz    a, -8(sp)
lwz    b, -4(sp)
stw    tos, -8(sp)
stw    a, -4(sp)
stw    b, 0(sp)
addi sp, sp, 4
used6
// 92 dup2
lwz    a, -4(sp)
stw    tos, 0(sp)
stw    a, 4(sp)
addi sp, sp, 8
used4
// 93 dup2_x1
lwz    a, -8(sp)
lwz    b, -4(sp)
stw    b, -8(sp)
stw    tos, -4(sp)
stw    a, 0(sp)
stw    b, 4(sp)
addi sp, sp, 8
used7
// 94 dup2_x2
lwz    a, -12(sp)
lwz    b, -8(sp)
lwz    c, -4(sp)
stw    c, -12(sp)
stw    tos, -8(sp)
stw    a, -4(sp)
stw    b, 0(sp)
b    dup2_x2_more
// 95 swap
lwz    a, -4(sp)
stw    tos, -4(sp)
mr   tos, a
used3

// 96 iadd
lwzu a, -4(sp)
add    tos, a, tos
used2
// 97 ladd
used6
// pretend we had 6
// instructions already
// stash the rest of
// dup2_x2 here
dup2_x2_more:
stw    c, 4(sp)
addi sp, sp, 8
used4
// 98 fadd
used0
// 99 dadd
used0
// 100 isub
lwzu a, -4(sp)
subf tos, tos, a
used2
// 101 lsub
used0
// 102 fsub
used0
// 103 dsub
used0
// 104 imul
lwzu a, -4(sp)
mullw  tos, tos, a
used2
// 105 lmul
used0
// 106 fmul
used0
// 107 dmul
used0
// 108 idiv
lwzu a, -4(sp)
divw tos, a, tos
used2
// 109 ldiv
used0
// 110 fdiv
used0
// 111 ddiv
used0
// 112 irem
lwzu a, -4(sp)
divw b, a, tos
mullw  c, b, tos
subf tos, c, a
used4
// 113 lrem
used0
// 114 frem
used0
// 115 drem
used0
// 116 ineg
neg    tos, tos
used1
// 117 lneg
used0
// 118 fneg
used0
// 119 dneg
used0
// 120 ishl
lwzu a, -4(sp)
andi.  tos, tos, 31
slw    tos, a, tos
used3
// 121 lshl
lwzu tos, -4(sp)
used1
// 122 ishr
lwzu a, -4(sp)
andi.  tos, tos, 31
sraw tos, a, tos
used3
// 123 lshr
lwzu tos, -4(sp)
used1
// 124 iushr
lwzu a, -4(sp)
andi.  tos, tos, 31
srw    tos, a, tos
used3
// 125 lushr
lwzu tos, -4(sp)
used1
// 126 iand
lwzu a, -4(sp)
and    tos, tos, a
used2
// 127 land
used0
// 128 ior
lwzu a, -4(sp)
or   tos, tos, a
used2
// 129 lor
used0
// 130 ixor
lwzu a, -4(sp)
xor    tos, tos, a
used2
// 131 lxor
used0
// 132 iinc
lhz    b, 0(ip)
rlwinm a, b, 26,
16, 29
extsb  b, b
addi ip, ip, 2
lwzx c, fp, a
add    c, c, b
stwx c, fp, a
used7
// 133 i2l
lwzu tos, -4(sp)
used1
// 134 i2f
lwzu tos, -4(sp)
used1
// 135 i2d
lwzu tos, -4(sp)
used1
// 136 l2i
pushi(0)
used3
// 137 l2f
used0
// 138 l2d
used0
// 139 f2i
pushi(0)
used3
// 140 f2f
used0
// 141 f2d
used0
// 142 d2i
pushi(0)
used3
// 143 d2f
used0
// 144 d2d
used0
// 145 int2byte
extsb  tos, tos
used1
// 146 int2char
andi.  tos, tos,
0xffff
used1
// 147 int2short
extsh  tos, tos
used1
// 148 lcmp
pushi(0)
used3
// 149 fcmpl
pushi(0)
used3
// 150 fcmpg
pushi(0)
used3
// 151 dcmpl
pushi(0)
used3
// 152 dcmpg
pushi(0)
used3
// 153 ifeq
cmpi cr0, 0, tos, 0
lha    a, 0(ip)
lwzu tos, -4(sp)
addi ip, ip, 2
subi a, a, 3
bne    next0
add    ip, ip, a
used7
// 154 ifne
cmpi cr0, 0, tos, 0
lha    a, 0(ip)
lwzu tos, -4(sp)
addi ip, ip, 2
subi a, a, 3
beq    next0
add    ip, ip, a
used7
// 155 iflt
cmpi cr0, 0, tos, 0
lha    a, 0(ip)
lwzu tos, -4(sp)
addi ip, ip, 2
subi a, a, 3
bge    next0
add    ip, ip, a
used7
// 156 ifge
cmpi cr0, 0, tos, 0
lha    a, 0(ip)
lwzu tos, -4(sp)
addi ip, ip, 2
subi a, a, 3
blt    next0
add    ip, ip, a
used7
// 157 ifgt
cmpi cr0, 0, tos, 0
lha    a, 0(ip)
lwzu tos, -4(sp)
addi ip, ip, 2
subi a, a, 3
ble    next0
add    ip, ip, a
used7
// 158 ifle
cmpi cr0, 0, tos, 0
lha    a, 0(ip)
lwzu tos, -4(sp)
addi ip, ip, 2
bgt    next0
// shared by cmp instructions
do_jump:
subi a, a, 3
add    ip, ip, a
used7
// 159 if_icmpeq
lwz    b, -4(sp)
cmp    cr0, 0, b,
tos
lha    a, 0(ip)
lwzu tos, -8(sp)
addi ip, ip, 2
beq    do_jump
used6
// 160 if_icmpne
lwz    b, -4(sp)
cmp    cr0, 0, b,
tos
lha    a, 0(ip)
lwzu tos, -8(sp)
addi ip, ip, 2
bne    do_jump
used6
// 161 if_icmplt
lwz    b, -4(sp)
cmp    cr0, 0, b,
tos
lha    a, 0(ip)
lwzu tos, -8(sp)
addi ip, ip, 2
blt    do_jump
used6
// 162 if_icmpge
lwz    b, -4(sp)
cmp    cr0, 0, b,
tos
lha    a, 0(ip)
lwzu tos, -8(sp)
addi ip, ip, 2
bge    do_jump
used6
// 163 if_icmpgt
lwz    b, -4(sp)
cmp    cr0, 0, b,
tos
lha    a, 0(ip)
lwzu tos, -8(sp)
addi ip, ip, 2
bgt    do_jump
used6
// 164 if_icmple
lwz    b, -4(sp)
cmp    cr0, 0, b,
tos
lha    a, 0(ip)
lwzu tos, -8(sp)
addi ip, ip, 2
ble    do_jump
used6
// 165 if_acmpeq
lwz    b, -4(sp)
cmp    cr0, 0, b,
tos
lha    a, 0(ip)
lwzu tos, -8(sp)
addi ip, ip, 2
beq    do_jump
used6
// 166 if_acmpne
lwz    b, -4(sp)
cmp    cr0, 0, b,
tos
lha    a, 0(ip)
lwzu tos, -8(sp)
addi ip, ip, 2
bne    do_jump
used6
// 167 goto
lha    a, 0(ip)
subi   ip, ip, 1
add    ip, ip, a
used3
// 168 jsr
pushtos
lha    a, 0(ip)
addi tos, ip, 2
// addr of next instr
subi ip, ip, 1
add    ip, ip, a
used6
// 169 ret
lbz    a, 0(ip)
times4(a)
// convert to offset
lwzx ip, fp, a
used3
// 170 tableswitch
mr   r3, ip
mr   r4, tsBase
mr   r5, tos
bl   TableSwitch
mr   ip, r3
poptos
```

```
used6
// 171 lookupswitch
mr   r3, ip
mr   r4, tsBase
mr   r5, tos
bl   LookupSwitch
mr   ip, r3
poptos
used6
// 172 ireturn
lwz    ip, 8(rp)
mr   sp, fp
lwz    fp, 4(rp)
lwz    tsBase,
0(rp)
addi rp, rp, 12
used5
// 173 lreturn
lwz    ip, 8(rp)
mr   sp, fp
lwz    fp, 4(rp)
lwz    tsBase,
0(rp)
lwzu tos, -4(sp)
addi rp, rp, 12
used6
// 174 freturn
lwz    ip, 8(rp)
mr   sp, fp
lwz    fp, 4(rp)
lwz    tsBase,
0(rp)
lwzu tos, -4(sp)
addi rp, rp, 12
used6
// 175 dreturn
lwz    ip, 8(rp)
mr   sp, fp
lwz    fp, 4(rp)
lwz    tsBase,
0(rp)
lwzu tos, -4(sp)
addi rp, rp, 12
used6
// 176 areturn
lwz    ip, 8(rp)
mr   sp, fp
lwz    fp, 4(rp)
lwz    tsBase,
0(rp)
addi rp, rp, 12
used5
// 177 return
lwz    ip, 8(rp)
mr   sp, fp
lwz    fp, 4(rp)
lwz    tsBase,
0(rp)
lwzu tos, -4(sp)
addi rp, rp, 12
used6
// 178 getstatic
lhz    a, 0(ip)
addi ip, ip, 2
times4(a)
lwzx b, cp, a
b    getstatic_more
nop;nop;nop



// 179 putstatic
lhz    a, 0(ip)
addi ip, ip, 2
times4(a)
lwzx b, cp, a
b    putstatic_more
nop;nop;nop
// 180 getfield
lhz    a, 0(ip)
addi ip, ip, 2
times4(a)
lwzx b, cp, a
b    getfield_more
nop;nop;nop
// 181 putfield
lhz    a, 0(ip)
addi ip, ip, 2
times4(a)
lwzx b, cp, a
b    putfield_more
nop;nop;nop
// 182 invokevirtual
lhz    a, 0(ip)
addi ip, ip, 2
times4(a)
// convert to index
lwzx a, cp, a
lwz    methodP,
1(a)
stwu ip, -4(rp)
b    start_method
nop
// 183 invokenonvirtual
lhz    a, 0(ip)
addi ip, ip, 2
times4(a)
// convert to index
lwzx a, cp, a
lwz    methodP,
1(a)
stwu ip, -4(rp)
b    start_method
nop
// 184 invokestatic
lhz    a, 0(ip)
addi ip, ip, 2
times4(a)
// convert to index
lwzx a, cp, a
lwz    methodP,
1(a)
stwu ip, -4(rp)
b    start_method
nop
// 185 invokeinterface
addi ip, ip, 2
used1
// 186 undefined
used0
// 187 new
addi ip, ip, 2
used1
// 188 newarray
lbz    r3, 0(ip)
addi ip, ip, 1
mr   r4, tos
bl   GetNewArray
mr   tos, r3
used5
// 189 anewarray
addi ip, ip, 2
// skip type
mr   r3, tos
bl   GetANewArray
mr   tos, r3
used4
// 190 arraylength
lwz    tos, 0(tos)
used1
// 191 athrow
used0
// 192 checkcast
addi ip, ip, 2
used1
// 193 instanceof
addi ip, ip, 2
li   tos, 1
// assume true
used2
// 194 monitorenter
poptos
used1
// 195 monitorexit
poptos
used1
// 196 wide
addi ip, ip, 1
used1
// 197 multianewarray
lhz    r3, 0(ip)
lbz    r4, 2(ip)
addi ip, ip, 3
pushtos
b
multianewarray_more
nop;nop;
// 198 ifnull
cmpi cr0, 0, tos, 0
lha    a, 0(ip)
lwzu tos, -4(sp)
addi ip, ip, 2
subi a, a, 3
bne    next0
add    ip, ip, a
used7
// 199 ifnonnull
cmpi cr0, 0, tos, 0
lha    a, 0(ip)
lwzu tos, -4(sp)
addi ip, ip, 2
subi a, a, 3
beq    next0
add    ip, ip, a
used7
// 200 goto_w
lwz    a, 0(ip)
subi a, a, 1
add    ip, ip, a
used3
// 201 jsr_w
pushtos
lwz    a, 0(ip)
addi tos, ip, 5
subi a, a, 1
add    ip, ip, a
used6
// 202 breakpoint
used0
// 203 unused - but we use
// it to signal final exit
b  exitVM
used1
used0     // 204
used0     // 205
used0     // 206
used0     // 207
used0     // 208
// 209 ret_w
lhz    a, 0(ip)
times4(a)
lwzx ip, fp, a
used3

// remaining opcodes unused

exitVM:
  mr   r3, tos
  frfree
  blr


ldc_more:
  stw    sp, SP
  bl   PushConstant
  lwz    sp, SP
  poptos
  b    next0

multianewarray_more:
  stw    sp, SP
  bl
PushMultiANewArray
  lwz    sp, SP
  poptos
  b    next0

getstatic_more:
  lwz    a, 1(b)
// get fieldP
  lwz    b, 8(a)
// get size
  cmpi cr0, 0, b, 0
  lwz    c, 4(a)
// get offset
  beq    next0
  pushtos
  lwzx tos, h0, c
  b    next0

putstatic_more:
  lwz    a, 1(b)
// get fieldP
  lwz    b, 8(a)
// get size
  cmpi cr0, 0, b, 0
  lwz    c, 4(a)
// get offset
  beq    drop1ngo
  stwx tos, h0, c
  poptos
  b    next0

getfield_more:
  lwz    a, 1(b)
// get fieldP
  lwz    b, 8(a)
// get size
  cmpi cr0, 0, b, 0
  lwz    c, 4(a)
// get offset
  beq    drop1ngo
  lwzx tos, tos, c
  b    next0

putfield_more:
  lwz    a, 1(b)
// get fieldP
  lwz    b, 8(a)
// get size
  cmpi cr0, 0, b, 0
  lwz    c, 4(a)
// get offset
  beq    drop2ngo
  lwzu a, -4(sp)
  stwx tos, a, c
  poptos
  b    next0

drop2ngo:
  poptos
drop1ngo:
  poptos
  b    next0

}
```

PushConstant

```
static void
PushConstant(long n)
{
  u1   *p;

  p = Constants[n];
  switch (*p++)
  {
    case C_Utf8:
    case C_Unicode:

  // since no operations act
  // on these objects,
  // I just make the const
  // data be the object
  // (w/o the tag) for
  // simplicity of the test
  // code.
      PUSH((long)p);
      break;
    case C_Integer:
      PUSH(*(s4*)p);
      break;
    case C_Float:
      break;
    case C_Long:
      break;
    case C_Double:
      break;
    case C_Class:
      break;
    case C_String:
    PushConstant
        (*(u2*)p);
      break;
    case C_Fieldref:
    case
C_Methodref:
    case
C_InterfaceMethodref:
    case
C_NameAndType:
        DebugStr
("\pit can happen");
      break;
  }
}
```

TableSwitch

```
static u1
*TableSwitch(u1 *ip,
long tsBase, long n)
{
  long *base;
  long defaultOffset;
  long low;
  long high;

  base = (long *)
(((((long)ip-tsBase)
+ 3) & -4)+tsBase);
  defaultOffset =
*base++;
  low = *base++;
  high = *base++;
  ip -= 1;

  if (n < low || n >
high)
    ip +=
defaultOffset;
  else
    ip += base[n -
low];
  return ip;
}
```

LookupSwitch

```
static u1
*LookupSwitch(u1 *ip,
long tsBase, long
key)
{
  long *base;
  long defaultOffset;
  long numPairs;
  long match;
    base = (long *)
(((((long)ip-tsBase)
+ 3) & -4)+tsBase);
  defaultOffset =
*base++;
  numPairs = *base++;
  ip -= 1;

  while (numPairs >
0)
  {
    match = *base++;
    numPairs -= 1;
    if (key ==
match)
      return ip +
*base;
    base += 1;
  }
  return ip +
defaultOffset;
```

```
}
```

GetNewArray

```
// Array object: # elements, data
// data is padded to multiple of 4
static long GetNewArray(long type, long size)
{
  long   result;
  long   elementSize;

  result = (long) HP;
  *HP++ = size;
  if (type == 10)
  // int
    elementSize = 4;
  else if ((type & 3) == 0)
  // boolean or byte
    elementSize = 1;
  else if ((type & 3) == 1)
  // char or short
    elementSize = 2;
  else
    elementSize = 0;
  // unsupported types
  HP += ((size * elementSize + 3) >> 2);
  return result;
}
```

GetANewArray

```
static long GetANewArray(long size)
{
  long   result;
  long   elementSize;
  u4     *objectP;

  result = (long) HP;
  *HP++ = size;
  elementSize = 4;
  objectP = (u4*) HP;
  HP = (long *) ((u1 *) HP + size * 4);
  while (size > 0)
  {
    size -= 1;
    *objectP = (u4)HP;
    HP += elementSize;
  }
  return result;
}
```

PushMultiANewArray

```
// Rather complicated function to allocate a
// multi-dimensional array as arrays of array objects.
static void PushMultiANewArray(long classIndex,
                  long numDimensions)
{
  u1   *cp;
  long nameIndex;
  u1   *name;
  int    type;
  long elementSize;
  long arrayRef;
  long size;
  long copies;
  long i, j;
  long dim;
  long dataLongs;
  long *subArrayRef;
  long subArrayLongs;

  arrayRef = (long) HP;
  cp = Constants[classIndex];
  nameIndex = *(u2*) (cp+1);
  cp = Constants[nameIndex];
  name = cp + 3;
  type = name[numDimensions];
  // skip the known '[' chars
  if (type == 'I' || type == '[')
  // int or ref
    elementSize = 4;
  else if (type == 'Z' || type == 'B')      // bool or byte
    elementSize = 1;
  else if (type == 'C' || type == 'S')      // char or short
    elementSize = 2;
  else
    elementSize = 0;  // unsupported types
  copies = 1;

  size = *(SP-1);      // the nth dimension
  dataLongs = (size * elementSize + 3) >> 2;
  for (dim = 0; dim < numDimensions - 1; dim++)
  {
    size = *(SP-numDimensions+dim);
    if (dim == numDimensions - 2)
      subArrayLongs = dataLongs + 1;
    else
      subArrayLongs = *(SP-numDimensions+dim+1) + 1;
    subArrayRef = HP + (size + 1) * copies;
    for (i = 0; i < copies; i++)
    {
      *HP++ = size;
      for (j = 0; j < size; j++)
      {
        *HP++ = (long) subArrayRef;
        subArrayRef += subArrayLongs;
      }
    }
    copies *= size;
  }
  // last dim is special since it has no subarrays
  size = *(SP-1);    // the nth dimension
  for (i = 0; i < copies; i++)
  {
    *HP++ = size;
    HP += dataLongs;
  }
  SP -= numDimensions;
  // remove dimensions
  PUSH(arrayRef);
}
```

BOOK EXCERPT

By Tony Francis

Mac OS 8 Address Spaces and Memory Protection

[As you may have heard by now, Apple has made the announcement that Mac OS 8 (aka Copland) will be delivered in parts instead of one large release. The first of these releases is slated for January, 1997. Some Mac OS 8 technologies previously announced may not ever be part of any release — but many are and will be implemented over the next 18 months. This month, we're bringing you an excerpt from Mac OS 8 Revealed by Addison-Wesley. This book contains important information/background about technologies that are definitely to come in one release or another. We at MacTech felt is was important for you to see what is coming so that you could intelligently plan for, discuss, and debate these new parts of the operating system. If you'd like to know more about other technologies, Mac OS 8 Revealed is a good source of information — just be aware of which technologies are coming in which timeframes (and if at all) when you do your planning. – Ed. nst]

When a program is launched—for instance, when a user double-clicks its icon—the operating system prepares the program code for execution, creates memory areas for the code and its temporary data, and assigns locations for the code and data within these memory areas. In this way, the program becomes instantiated as a process on the computer. The memory areas created for a process lie within a 4-gigabyte (GB) range of logical addresses. This range of addressable memory constitutes the address space for that process.

Mac OS 8 maintains multiple simultaneous address spaces. A program can't reference any memory locations outside of its address space. Therefore, if code in a given address space malfunctions, it can't corrupt the data in a different address space. Mac OS 8 provides other forms of memory protection, too. Mac OS 8 protects all code, for example, by mapping it into read-only memory areas where it can't be corrupted by any errant code elsewhere in the system. Crucial system data is protected because it's stored in memory areas where operating system services—such as the microkernel, device drivers, and the file system—have read/write permission to the data, but application-level software has read-only permission. This greatly decreases the ability of applications to cause a system-wide crash. Yet another kind of memory protection, called **guard pages**, enhances system stability by limiting the amount of damage that software can do if it attempts to read or write outside the memory area it's entitled to access.

Key Terms and Concepts

- A **process** is an instance of a program running at execution time. A process is characterized by a set of one or more tasks and the operating system resources necessary to support those tasks.
- A **task** is the basic unit of program execution in Mac OS 8. Every process has at least one task. As you'll read in the next chapter, each task is assigned a priority and, when eligible for execution, is preemptively scheduled by the microkernel.
- A **memory area** is a range of logical addresses.
- **Virtual memory** is addressable memory beyond the limits of available physical memory. Mac OS 8 extends physical memory by storing on a secondary storage device, such as a hard disk, code and data not immediately required by the CPU.
- A **logical address** is a memory address used by code when it's running. By comparison, a **physical address** is a

memory address represented by bits on a physical address bus. Physical addresses are assigned to memory locations in RAM chips and to various hardware devices. When executing code, the CPU translates the logical addresses of an address space into physical addresses.

- An access **permission** stipulates whether other programs can read from or write to a memory area.
- A **guard page** is a 4-kilobyte (K) range of logical addresses that excludes all program access. Guard pages may appear at the beginnings and ends of memory areas to help prevent code from inadvertently accessing the wrong memory areas. If a programming error causes code to reference a guard page, the CPU generates an exception before the erring code can adversely affect a contiguous memory area.

Major Points of Interest

All code and data for a process exist within an address space. Because Mac OS 8 uses a 32-bit address space—which is the maximum size supported by the PowerPC CPU—an address space can contain up to 232 addresses. In every address space, in other words, addressable locations number up to 4GB.

A 4-GB address space encompasses far more memory addresses than are available in physical memory on most computers. So Mac OS 8 uses a virtual memory system to extend the range of addressable memory beyond what is available in physical memory. The virtual memory system stores unused portions of code and data on a secondary storage device, such as hard disk. The virtual memory system then transfers into physical memory only those portions immediately needed by the CPU. (As you'll see in Chapter 6, the virtual memory system also makes efficient use of secondary storage by using only enough disk space to support currently open programs.)

When launching a program, the operating system creates memory areas that constitute only a small portion of an address space. The operating system creates a memory area for the program code, and it creates an initial memory area for the program to store the data—such as its global variables and dynamic data structures—that it needs while it's running. Other portions of an address space are unavailable to the program because they're used to store code (including code for the microkernel and code for the libraries used by the program), or they're reserved for other uses by the operating system. From the 4GB of logical addresses in a single address space, at least 1GB is available to programs for data storage.

As you'll see in Chapter 7, the operating system dynamically creates and releases memory areas as needed so that programs can store temporary data. The Dynamic Storage-Allocation Services provided by Mac OS 8 also allow developers to create their own memory areas suitable for special program needs.

For overall system stability, Mac OS 8 employs multiple address spaces. The data referenced by a program in one address space is inaccessible to programs in other address spaces. Therefore, programming errors affecting one address space are isolated from all other address spaces. For example, suppose that a game program has a programming error that corrupts portions of its address space, causing the game to crash. Operating on data in its own address space, a World Wide Web server program continues serving web pages, immune to the game's error.

Within an address space, areas of memory may be further protected by access permissions. For example, all executable code in Mac OS 8 is stored in read-only memory areas where code can't possibly be corrupted. And data used by critical portions of the operation system, such as the microkernel, is kept in areas protected by access permissions that prevent applications from corrupting it.

For compatibility with System 7 applications, which rely on a single address space, all cooperative programs share a single address space. Every server program, by comparison, is given its own address space.

The Cooperative Program Address Space

Whereas Mac OS 8 supports multiple address spaces, System 7 supports only one address space. To provide compatibility for System 7 applications, many of which are designed to read or manipulate each other's data structures, Mac OS 8 assigns all cooperative programs to a shared address space. Figure 3.1 illustrates the cooperative-program address space for a system on which the user has launched an e-mail editing program and a game program from the Finder program. All three cooperative programs store their temporary data in this address space. (These applications, by the way, are cooperative programs because they present a human interface.)

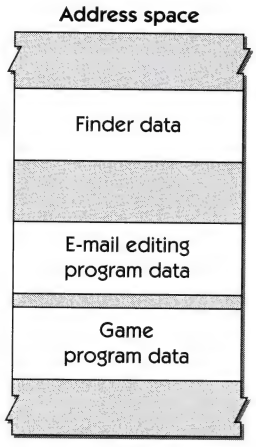


Figure 3.1: Cooperative programs sharing an address space

The figures in this book don't literally represent the layout of logical memory. For example, data for the Finder appears near the top of the address space in Figure 3.1; however, Finder data isn't necessarily mapped into memory areas at the top of the cooperative address space.

Whereas the amount of memory that's available to applications in System 7 is usually far less than 4GB, an entire 4-GB address space is available to them in Mac OS 8. This large amount of addressable memory, backed by the Mac OS 8 virtual

World Wide Web Weaver For Macintosh And Power Macintosh

A professional, powerful, and easy to use HTML editor for Macintosh and PowerMacintosh

World Wide Web Weaver

NEW VERSION 2.0

The professional web page development tool

- Interactive SpellChecker
- True, continuously updated, WYSIWYG preview
- Tables (creating, editing, and importing)
- Limited site management
- Forms
- Page colors
- Frames
- Text and styled text importing from word processors and others
- Add your own tags
- Very customizable
- Preview with different web browsers
- Includes two free upgrades

World Wide Web Weaver, the professional World Wide Web page creation tool, is an application designed to help you create World Wide Web pages using an easy yet flexible interface. World Wide Web Weaver gives a true WYSIWYG preview of your document, which is updated as you type, allows you access to all HTML, yet does not expect you type any HTML yourself (unless you want to!)

"Miracle Software's World Wide Web Weaver 1.1 is a simple, usefull HTML editor that helps authors think visually while editing HTML tags."

MacUser, August 1996

$89 - Direct From Miracle Software	(315) 265-0930
$79 - From Mac Zone	(800) 248-0800
$79 - From Publisher's Toolbox	(800) 390-0461

$60 - Educational price (direct only)

$35 - Upgrade from World Wide Web Weaver 1.x (direct only)

Call for Multi-User Packs

For the latest ordering information point your web browser to http://www.MiracleInc.com or call (315) 265-0930.

memory system, allows the user to keep many more applications open simultaneously than is possible in System 7.

Like Mac OS 8, System 7 uses a 32-bit address space, where any address between 0x0000 0000 and 0xFFFF FFFF is a valid logical address. In System 7, however, the range of logical addresses actually available from this address space is determined at system startup by the amount of virtual memory previously selected by the user. Mac OS 8, by comparison, dynamically allocates storage locations from this address range to satisfy program needs as they arise.

For example, if a user in System 7 sets total memory to 12MB and launches an e-mail application and a game, they'd share 12MB of addressable memory even if they required only 5MB between them. If the user then tried to launch a photo-editing application requiring 8MB of addressable memory, the program would fail to open because of insufficient memory. To launch the photo-editing program, the user would need to quit the e-mail application or the game.

When these same programs are launched in Mac OS 8, the operating system supplies their memory needs dynamically. For example, the operating system allocates from the 4-GB address space only the 5MB necessary to run the e-mail program and the game. When the user launches the photo-editing application, the operating system allocates another 8MB from this address space. As the user launches more applications, Mac OS 8 continues allocating more addressable memory from the address space. (As you'll see in Chapter 6, the number and size of applications that the user may launch are constrained only by the disk space available to the virtual memory system for storing temporary data. To extend virtual memory without consuming any additional disk space, the operating system memory-maps the disk files of all code used at execution time.)

The enormous range of addressable memory that Mac OS 8 supplies to cooperative programs nearly eliminates the memory fragmentation problems experienced by users of operating systems supplying smaller amounts of addressable memory. For example, a System 7 user might launch enough applications to fill all 12MB of available memory and then quit two applications to release 8MB of memory. If the two applications weren't contiguous in memory, the total available memory might be fragmented into two 4-MB areas, preventing the user from launching a 5-MB application. On a Mac OS 8 system, memory for this application would be allocated from some unused portion of the 4-GB address space.

Protected Address Spaces for Server Programs

When a server program is launched (usually this happens automatically when the user starts the computer), the operating system instantiates the process for that server program in its own address space. Because every server program exists in its own address space, where other programs can't address its data, server programs are protected from possible programming errors in cooperative programs and other server programs.

Figure 3.2 illustrates separate address spaces for two server programs: an e-mail server program and a World Wide Web server program. Each program operates on data stored exclusively in its own address space.

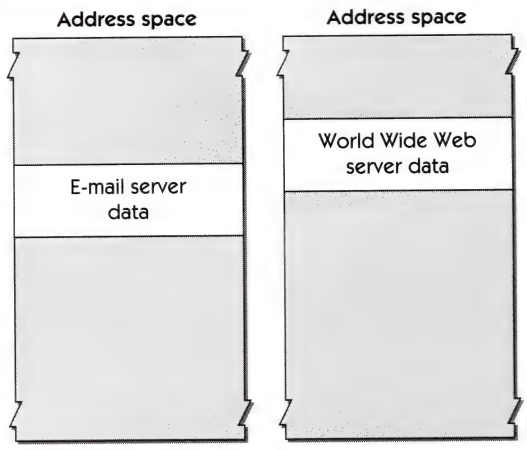


Figure 3.2: Server programs protected by separate address spaces

To protect a program from being corrupted by other programs, a developer can implement portions of an application as a server program. Only the portions of an application that incorporate a human interface need to be implemented in a cooperative program. For example, after a user writes an electronic mail message with an e-mail editing program, that cooperative program can call an e-mail server program and request the server program to deliver the message over a network. Likewise, the e-mail server program can receive messages sent to the user from across the network and store them until the user is ready to read them with the e-mail editing program.

To protect critical system data and increase system reliability, many nonprivileged Mac OS 8 services are implemented as server programs. For example, the Process Manager and the Font Manager (which provides font-rendering services to the system) are implemented as server programs, each in its own protected address space. As you'll see later in this chapter, privileged code—such as the microkernel—has protection mechanisms of its own.

Another benefit to designing software as a server program is that it has an address space all to itself for storing its temporary data. Cooperative programs, by contrast, must share their address space with each other, reducing the amount of address space available to each cooperative program.

Address Space Switching by the Microkernel

The CPU can read from and write to the memory of only one address space at a time. The microkernel is responsible for keeping track of all the memory addresses for the code and data residing in these address spaces. The microkernel manages these address spaces so that the CPU works with only one address space at a time.

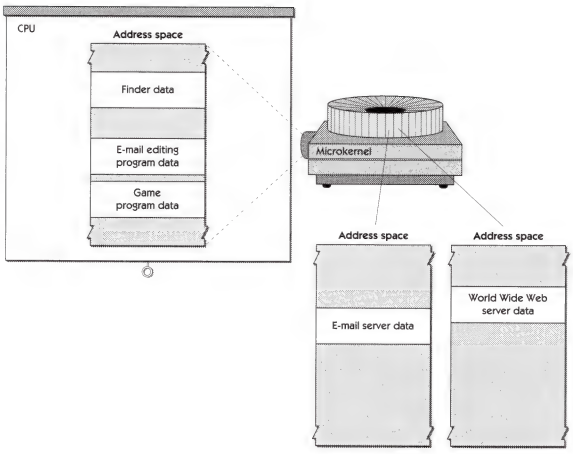


Figure 3.3: Switching between address spaces

Figure 3.3 symbolizes how the microkernel manages multiple address spaces. In this figure, address spaces are represented as slides in a slide projector. The microkernel operates like the slide projector—while many address spaces are available, the microkernel projects only one at a time onto the CPU. In this figure, the microkernel is projecting the cooperative program address space onto the CPU, represented here as a projection screen. When the microkernel determines that it's time for one of the server programs to execute on the CPU, the microkernel "projects" that program's address space onto the CPU. (Chapter 4 explains how the operating system determines which task of which program gets to execute on the CPU at any given moment.)

System-Wide and Shared Memory Areas

A memory area is a range of logical addresses within an address space. In addition to supporting memory areas specific to individual address spaces, Mac OS 8 also maintains

- **system-wide memory** areas, which can be referenced across all address spaces
- **shared memory areas**, which can be referenced within two or more address spaces

A system-wide memory area appears at the same location in every address space. The contents of a system-wide area are potentially visible in all address spaces. For example, the microkernel employs system-wide memory areas for storing its own data, as shown in Figure 3.4. The microkernel is essentially a process that exists simultaneously in every address space. By storing its data in system-wide memory areas, the microkernel can efficiently manage system-wide responsibilities. (To protect the stability of the entire system, only other essential operating system services—such as device drivers—have permission to change the data in the microkernel's system-wide memory areas. Access permissions are described in the next section.)

The operating system also maps all executable code into system-wide memory areas. Thus, a single copy of the code from any library—such as any of the libraries implementing operating system services—can be efficiently shared by all of the programs using that library. As Figure 3.4 illustrates, the code for all programs on a system exists in identical locations across all address spaces in the system, even though the programs store their data in memory areas local to each address space.

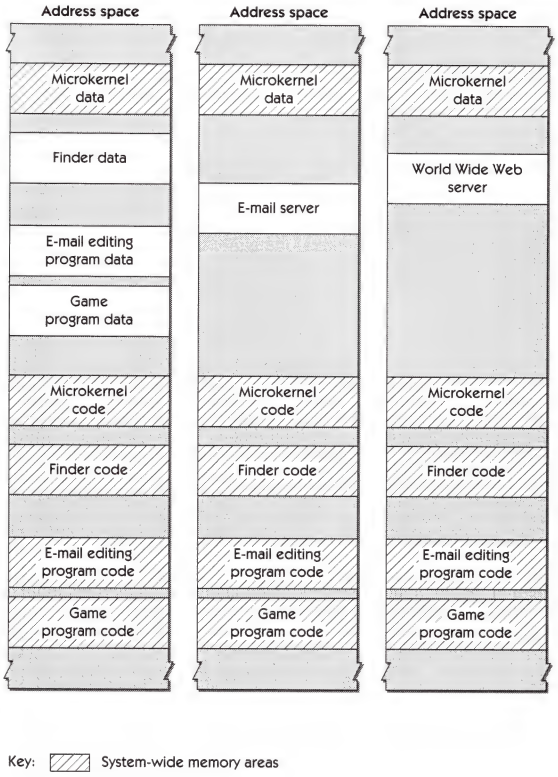


Figure 3.4: System-wide memory areas

A program can create a system-wide memory area to share its data with programs in other address spaces. More likely, however, a program will use a shared memory area for this purpose. A shared memory area exists in two or more address spaces, but not necessarily all address spaces. A shared memory area can begin at the same address in various address spaces (which is useful if shared data is accessed by pointers, because pointers contain memory addresses), or it can begin at different addresses. A shared memory area can have different access permissions in different address spaces. For example, a program can write data into a shared memory area in its own address space but, as you'll see in the next section, make the data read-only to programs in other address spaces, thereby granting other programs access to a reliable copy of the data.

Additional Forms of Memory Protection

You've seen how Mac OS 8 separates server programs into their own address spaces, making them and the entire system more reliable. In addition to the protection afforded by separate address spaces, Mac OS 8 offers two more levels of memory protection that reduce the possibility of

one program corrupting the code or data used by another:

- access permissions for memory areas
- guard pages for memory areas

Access Permissions for Memory Areas

Access permissions provide additional protection to memory areas, even to those within a single address space. A program can create a memory area and set one of these three permission levels:

- **read/write**, which allows tasks in the same address space to view and change the contents of the memory area
- **read-only**, which allows tasks in the same address space to view but not change the contents of the memory area
- **excluded**, which forbids all tasks from reading from and writing to the memory area

When a program or the operating system assigns either read-only or excluded permission to a memory area, its contents are safe from corruption from other programs because no other program can write to that memory area. If a program or the operating system attempts to access a memory area to which it has insufficient access privileges, the processor generates an exception. An **exception** is an error or other special condition that is detected by the CPU during code execution. An exception transfers control from the code generating the exception to another piece of code, usually an exception handler.

As you've seen, the operating system maps all executable code into system-wide memory areas. These areas are assigned read-only permission, thereby preventing any program from writing over and corrupting the code of any other program.

If a program needs to share data with other programs, it can create a read-only memory area for the data. The creator of a memory area can also specify separate access permissions for nonprivileged and privileged code. **Nonprivileged code** is executed while the CPU is in user mode. **User mode**, in turn, is a state of operation for the PowerPC CPU that protects certain processor resources, such as various processor registers, from being modified. (Nonprivileged code is restricted from using various CPU instructions and hardware addresses and from changing data used by critical portions of the operating system. (To protect the stability of the user's system, most code in Mac OS 8 runs while the processor is in user mode.) A **processor register** is a named area of high-speed memory located on the CPU.)

Only the code for device drivers, the microkernel, and some other portions of the operating system is privileged. **Privileged code** is executed while the CPU is in supervisor mode. **Supervisor mode**, in turn, is a state of operation for the PowerPC CPU that allows full access to critical processor resources, such as all processor instructions and the tables that control memory protection. Privileged code can execute CPU instructions that are restricted from nonprivileged code and can access hardware addresses invisible to nonprivileged code.

The data used by privileged code can be excluded from nonprivileged code. A device driver, for example, may create a memory area that allows read/write access to privileged software but read-only access to nonprivileged software. Even privileged software can be denied write access to a memory area. For example, the system-wide memory areas containing code are always assigned read-only access for both privileged and nonprivileged software. Video RAM, which also resides in a system-wide memory area, is assigned read/write permission for both nonprivileged and privileged code.

(As a sidelight, it should be noted that to help protect system reliability, only privileged code can switch the CPU between supervisor mode and user mode. The microkernel always runs in supervisor mode; functions that call the microkernel cause the CPU to switch to supervisor mode. Before returning execution control back to nonprivileged code, the microkernel switches the CPU back to user mode.)

Guard Pages

A **page** is the smallest unit, measured in bytes, of information that the virtual memory system can transfer between physical memory and backing store. As you'll see in Chapter 6, a memory area is always a multiple of some number of pages.

Guard pages provide another level of protection, even to memory areas with read/write permission. When any program is launched in Mac OS 8, the operating system automatically places one or more guard pages at each end the program's stack and around the areas (sometimes known as **heaps**) created for its dynamic memory allocation needs. A program can specify its own number of guard pages to appear at the beginning and end of these areas and around any additional memory areas it creates. Mac OS 8 allows no access whatsoever to guard pages; neither privileged nor nonprivileged software can write to or read from them.

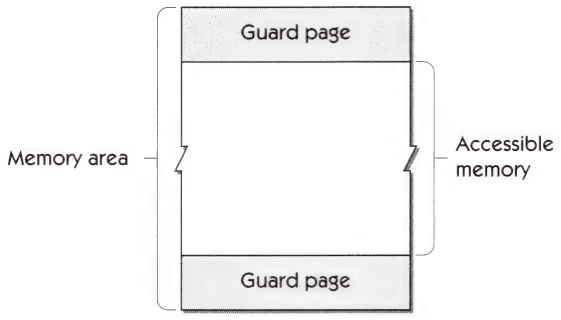


Figure 3.5: A memory area with guard pages

Figure 3.5 illustrates a memory area with guard pages. If any code, even for the program using that memory area, attempts to access a guard page, the CPU generates an exception. For example, a program can surround its stack with a range of guard pages equal to the length of its maximum stack frame. These guard pages then prevent the program's stack from overflowing into the memory area of any other

program. If the stack were to overflow and the stack attempted to access one of its guard pages, the CPU would send an exception to the program with the overflowing stack, resulting in the termination of that program before it could adversely affect any adjoining memory areas.

(A **stack** is a memory area where a task stores some of its temporary variables during execution. A **stack frame** is the area of the stack used by a routine for its parameters, return address, local variables, and temporary storage.)

Summary

Mac OS 8 uses multiple address spaces. The microkernel manages the system's multiple address spaces so that the CPU always references the right address space at the proper time.

By separating server programs into their own address spaces, Mac OS 8 protects these programs, making them and the whole system more reliable. Cooperative programs share a single address space to support System 7 application compatibility. Within this 4-GB address space, the large amount of addressable memory virtually eliminates memory fragmentation problems so that the user can open the greatest possible number of cooperative programs.

Mac OS 8 provides other forms of memory protection, too. First, programs as well as the operating system can assign read-only or excluded privileges to memory areas, thereby limiting access to and possible corruption of these areas by other programs. The operating system, for example, loads all code in areas that permit read-only access. Second, a program can place guard pages around a memory area to help prevent the program from accidentally accessing adjacent memory areas.

In order for code and data to be shared among address spaces, Mac OS 8 provides system-wide memory areas, which are visible in every address space, and shared memory areas, which are visible only in the address spaces of the programs that need access to these areas.

Planning a Product for Mac OS 8

If you're a developer, you can begin preparing to take advantage of multiple address spaces by determining whether some portion of your product benefits from the extra protection afforded by a separate address space. If so, you should plan to implement this portion as a server program.

MT

By Keith McGlauflin, Sidney, ME

Netscape Navigator Plug-ins

How to Enhance Navigator's Features

With the release of Netscape Communications' Navigator 2.0, dynamic code modules, called plug-ins, were introduced to seamlessly enhance Navigator's functionality without altering its user interface. This article contains four sections: an introduction to plug-ins and their installation, HTML tags for loading plug-ins, the plug-in Application Programming Interface (API), and a sample plug-in.

Plug-in Introduction and Installation

Plug-ins provide a way for third party developers to enhance Navigator to support other document types without requiring Navigator to launch helper applications on the user's Macintosh. Unlike Java applets, plug-ins are platform native, which means that a plug-in written for the Macintosh will not work on Windows or UNIX platforms. All Navigator plug-ins exist in the Plug-ins folder that is located in the same folder as Navigator.

Plug-ins can be used to communicate through AppleEvents to other software applications on the user's computer, or can take advantage of other Apple system software features, such as OpenDoc or QuickTime VR, without changing Navigator's source code.

Above all, plug-in implementation doesn't affect the overall user interface for Navigator, making navigation around the World-Wide Web (by clicks on links as well as the "Back" button and bookmarks) the same from the user's perspective even though a plug-in is loaded on the current page.

To install a new plug-in, simply drop the plug-in into the Plug-ins folder and restart Navigator if it is running. When Navigator starts up it scans the Plug-ins folder for files of type NSPL, the type used for plug-ins. Next, a list of MIME types and file name extensions handled by each plug-in is created so that Navigator can map the MIME types and extensions of documents referenced on HTML pages to plug-ins. For example, the MIME type application/myplug could be mapped to the extension .mypl for the plug-in myplugin which would cause any files with names that end with .mypl to load the plug-in (the HTML used to load plug-ins and how to specify MIME types and extensions for plug-ins are explained in the second and third sections of this article respectively). The MIME types for plug-ins are briefly displayed on Navigator's splash screen as it starts up. When Navigator loads a document with a name that ends with one of the extensions mapped to a MIME type for a plug-in, the plug-in is loaded by Navigator. The plug-in is unloaded when the user leaves the page which references the plug-in or when the user quits.

HTML Needed to Display Plug-ins

Plug-ins can be one of three types: full screen, embedded, or background. Full screen plug-ins are displayed in a window separate from HTML code, and are usually loaded by clicking on a link to a document with a name that ends with an extension that is mapped to a plug-in MIME type. Embedded plug-ins are included in the same screen as the HTML for the current page using the EMBED tag (the sample plug-in demonstrated in this article is an embedded plug-in). Background plug-ins are used to perform tasks which don't require user interaction, such as playing audio clips.

Keith McGlauflin is a Communications Specialist for Colby College. Part of his responsibilities include system management of the College's Web site, including writing CGIs and creating ways to use the Web in an educational environment. Keith can be contacted at kamcglau@colby.edu.

Full screen and background plug-ins can be loaded with a anchor tag that the user clicks on, such as:

```
<A HREF="http://foo.bar.com/testdoc.spec">
```

where .spec files are a plug-in extension .
For embedded plug-ins the **EMBED** tag is employed, such as:

```
<EMBED SRC="http://foo.bar.com/testdoc.spec"
PLUGINSPAGE="http://foo.bar.com/plugins/testdoc.html"
ALIGN=CENTER WIDTH=200 HEIGHT=75>
```

Where **SRC** is the URL to a document with a name that ends with of one of the extensions which is mapped to one of the MIME types handled by one of the plug-ins in the Plug-ins folder (in this case .spec). The optional **PLUGINSPAGE** tag gives the URL of a page which has documentation for this plug-in. The other options specify alignment and size, similar to the **IMG** tag. See the documentation which is included in the Plug-in Software Development Kit (SDK) for other optional attributes that can be used with the **EMBED** tag (the URL for downloading the SDK is given in the plug-in API section of this article).

Whichever method you use, loading a file with an extension that is mapped to a MIME type of a plug-in creates an "instance" of the plug-in. Instances are used to differentiate between references to the same plug-in using multiple data files with the same extension. This means you can have several **EMBED** tags which call the same plug-in but use different data files.

The PLUG-IN API

Netscape Communications provides three platform dependent SDKs (UNIX, Windows, Mac). The latest version of the SDK (version 3.0 at the time of this writing) is available at http://home.netscape.com/comprod/development_partners/plugin_api/index.html.

Version 3.0 of the Plug-in SDK doesn't include the very useful Plugin Template folder that was included with version 2. When creating plug-ins, it is much easier to start with a template and fill in those methods than it is to write the plug-in from scratch. The SDK does include several sample plug-ins that you can use as templates for your plug-in.

Mac SDK Methods

The Macintosh SDK includes CodeWarrior example projects which provide the method headers, resources, and C++ source code for a several sample plug-ins. You can either modify the source for one of these examples, or download the source for this article's sample plug-in and use it as your template for creating a plug-in. All you do is fill in the plug-in methods and write the subroutines which are called by those methods. Table 1 shows the method names which Navigator will call in your plug-in and the purpose of each method.

NPP_Initialize	Global initialization of the plug-in. Use to load resources shared by all plug-in instances.
NPP_Shutdown	Called when the last instance of the plug-in is destroyed.
NPP_New	Creates a new instance of the plug-in.
NPP_Destroy	Deletes an instance of a plug-in.
NPP_SetWindow	Assigns a window for the plug-in to draw into.
NPP_NewStream	Notifies the plug-in that a new data stream has been created.
NPP_WriteReady	Returns the maximum number of bytes the plugin can handle from a data stream.
NPP_Write	Reads data from a data stream and returns the number of bytes read.
NPP_DestroyStream	Indicates that a stream is to be destroyed.
NPP_StreamAsFile	Gives a local file for the data from a stream.
NPP_Print	Print an instance.
NPP_GetJavaClass	Returns the Java class of a plug-in.
NPP_URLNotify	Notifies the plug-in when a URL request completes.

Table 1: Plug-in Methods

Your plug-in must contain the routines listed in Table 1, even if you don't implement them fully (see **NPP_GetJavaClass** and **NPP_URLNotify** in the example plug-in's source code).

Also, there are methods available for your plug-in to call which cause Navigator to perform some action. These Navigator methods are beyond the scope of this article, but are documented in the Plug-in SDK's documentation.

Macintosh plug-ins also have methods which are Mac platform specific, as shown in Table 2.

NPP_HandleEvent	Used to handle an event received from Navigator.
NPN_MemAlloc	Allocate a block of memory
NPN_MemFree	Free up allocated memory

Table 2: Macintosh Specific Plug-in and NavigatorMethods

NPP_HandleEvent is a method that you must implement in your plug-in to handle user events (mouse clicks, etc.). **NPN_MemAlloc** and **NPN_MemFree** are methods that you don't implement in your plug-in, but, instead are available for your plug-in to call to allocate and free memory (they are Navigator methods).

The order in which Navigator calls the these methods in your plug-in is described in the sample plug-in section of this article.

Windows and Events

Several of these methods receive as parameters pointers to structures which hold the platform specific pointers to windows and events. Your methods will cast the generic pointers to the platform specific pointers your methods will need. For example, **NPP_HandleEvent**'s header looks like:

```
int16 NPP_HandleEvent(NPP instance, void* event)
```

and the **event** pointer can be cast into an EventRecord pointer:

```
EventRecord*ev;
ev = (EventRecord*) event;
```

Windows are a little more complicated but can be as easily cast into a WindowPtr or CGrafPtr. The structure for a generic window, NPWindow, looks like:

```
typedef struct _NPWindow
{
    void*   window;
    uint32  x;
    uint32  y;
    uint32  width;
    uint32  height;
    NPRect  clipRect;
} NPWindow;
```

and the pointer window points to an NP_Port:

```
typedef struct NP_Port
{
    CGrafPtr    port;
    int32       portx;
    int32       porty;
} NP_Port;
```

The CGrafPtr port is the pointer to the CGrafPort for the plug-in to draw to. Converting from the NPWindow record to a WindowPtr can be done as follows:

```
WindowPtr  theWindow;
NP_Port  *port;

port = ( NP_Port* ) window->window;
theWindow = ( WindowPtr ) port->port;
```

Creating a PLUG-IN: QuickTest

The sample plug-in, QuickTest, reads in a file of extension .test, converts its text to minimum, maximum, and current values, and displays an indicator. Two controls are also displayed, one to increment the indicator's current value and one to decrement the value (see figure 1).

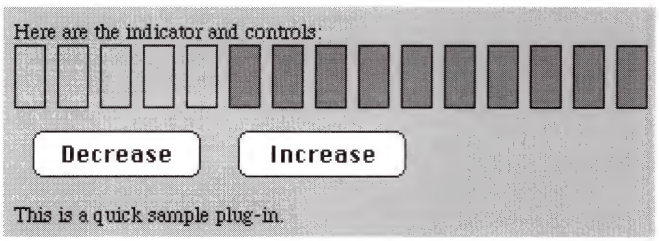


Figure 1: Screen shot of the QuickTest plug-in

To create the sample plug-in first download the Netscape Navigator Macintosh SDK at the URL listed above. If you don't have access to the MacTech Web site to download the project files for the sample plug-in you can easily create them using the example plug-ins included with the SDK.

First, create a new folder for the plug-in named QuickTest. Copy the project file from one of the example plug-ins to the QuickTest folder (e.g. copy PPViewPict68K.μ to QuickTest68K.μ).

Next, open up the project file for the platform you choose to implement (in this example I'll be using 68k code; PPC code is virtually the same). Remove the source code files from the

example's project window except for npmac.cp and the routines under the Glue section if you are implementing a PPC plug-in. Create QuickTest.rsrc with your favorite resource editor. Open up the QuickTest.rsrc file and create STR# 128 string 1 for the MIME type for QuickTest, application/test, and create string 2 for the extension for our plug-in files, .test (see figure 2). STR# 128 is used by Navigator to determine the extension/MIME type mapping for a plug-in. Next, create two PICT resources: one ID 3000 which is a light colored rectangle, and the other ID 3001 which is dark colored. Save QuickTest.rsrc and add it to the QuickTest project window.

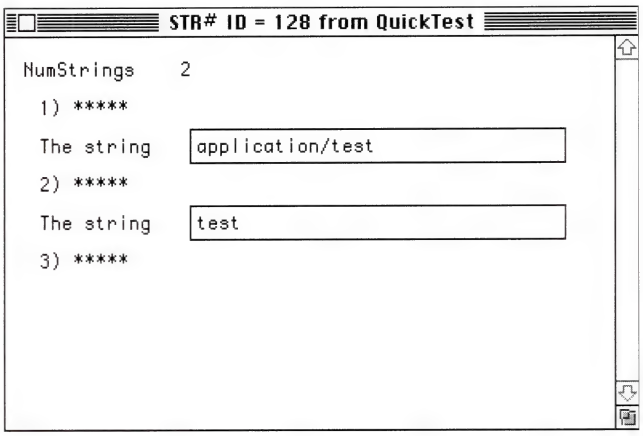


Figure 2: STR# 128 resource for QuickTest

Next, type in the code for the plug-in (see Listing 1) and save it as QuickTest.cp.

Listing 1: QuckTest.cp

```
// QuickTest.cp by Keith McGlauflin
// Based on npshell.cp by Netscape Communications
// Portions copyright 1996 Netscape Communications

// This Netscape Navigator plug-in displays an indicator
// strip based on the contents of the .test file referenced
// in an HTML file.  Controls are displayed to increase and
// decrease the indicator's level.

#ifndef _NPAPI_H_
#include "npapi.h"                  // Include Netscape's header file
#endif

#define  kMax 20                    // Maximum indicator value
#define  kMin 1                     // Minimum indicator value
#define kDecrease "\pDecrease"      // Decrease button text
#define kIncrease "\pIncrease"      // Increase button text
#define kBufferSize 1               // Size of the read buffer in bytes

// Plug-in Instance Data

typedef struct _Plug-inInstance
                                    // Data structure to hold all our variables
{                                   // for this plug-in instance:
  NPWindow*   fWindow;              // Netscape Plug-in Window record
  uint16    fMode;                  // Plug-in's mode (Full screen, embedded,
                                    // or background)
  Boolean   amBusy;                 // Are we loading data?
  Ptr         data;                 // Pointer to our data read from
                                    // the .test file
  int         datalength;           // Length of the data in Ptr
  Byte      min;                    // Indictator's minimum value
  Byte      max;                    // Indictator's maximum value
  Byte      current;                // Indictator's current value
  ControlHandle decControl;         // Handle to the Decrement control
  ControlHandle incControl;         // Handle to the Increment control
} Plug-inInstance;

// Our plug-in's globals:

CGrafPort gSavePort;                // Saved Port setting
CGrafPtr gOldPort;                  // Original Port settings
short   gRFRN;                      // Resource fork reference number
                                    // for this plug-in's resource fork
Handle  gonPict;                    // Handle to the indicator's ON pict
Handle  goffPict;                   // Handle to the indicator's OFF pict

// Method prototypes:

Boolean SavePort( NPWindow *window );
void   RestorePort( NPWindow *window );
void   DrawContents( Plug-inInstance *This );
void HandleContents( Plug-inInstance *This, Point where,
                WindowPtr theWindow );
void AddControls( Plug-inInstance *This );
void UpdateCntrls( NPWindow *window );
void   GetData( Plug-inInstance *This, unsigned long len, void
                *buffer );
void SetValues( Plug-inInstance *This );
void   SetDefaults( Plug-inInstance *This );
int    GetValue( Ptr bufferstart );
```

NPP_Initialize

```
// NPP_Initialize: (from npshell.cp) This procedure sets the clip region for our
// gSavePort global, loads the gonPict and goffPict ControlHandles, and returns
// NPERR_NO_ERROR.

NPError NPP_Initialize(void)
{
  gSavePort.clipRgn = ::NewRgn();

  gRFRN = CurResFile();                    // Get the plug-in's resource file

  if ( ResError( ) == noErr )
    {
    gonPict = GetResource( 'PICT', 3000 );
                                           // Get the ON pict
    goffPict = GetResource( 'PICT', 3001 );
                                           // Get the OFF pict
    }

  DetachResource( gonPict );
  DetachResource( goffPict );

  return NPERR_NO_ERROR;
}
```

NPP_Shutdown

```
// NPP_Shutdown: (from npshell.cp) This procedure disposes of our clip region in
// gSavePort and releases the indicator's ON and OFF pict handles.

void NPP_Shutdown(void )
{
  if (gSavePort.clipRgn)
    ::DisposeRgn(gSavePort.clipRgn);

  ReleaseResource( gonPict );      // Release the on Pict's resource
  ReleaseResource( goffPict );     // Release the off Pict's resource

}
```

NPP_New

```
// NPP_New: (from noshell.cp) This routine creates a new plug-in instance. First the
// plug-in validates the instance we received from Navigator, then it allocates enough
// memory to hold the PluginInstance data structure. Finally, the plug-in sets all
// the variables of our instance to their initial state.
```

```
NPError NPP_New(NPMIMEType plug-inType,
        NPP instance,
        uint16 mode,
        int16 argc,
        char* argn[],
        char* argv[],
        NPSavedData* saved)
{
  if (instance == NULL)
    return NPERR_INVALID_INSTANCE_ERROR;
              // Check for invalid plug-in instance

  instance->pdata = NPN_MemAlloc(sizeof(Plug-inInstance));
  Plug-inInstance* This = (Plug-inInstance*) instance->pdata;
  if (This != NULL)
  {                                  // Initialize our plug-in instance's variables
    This->fWindow = NULL;            // No window assigned yet
    This->amBusy = FALSE;            // Not currently loading data
    This->data = NULL;               // Data pointer is NULL
    This->datalength =0;             // Length of data is zero
    This->min = 0;                   // Min, max and current are set
    This->max = 0;                   // to zero
    This->current =0;
    This->decControl = NULL;         // ControlHandles set to NULL
    This->incControl = NULL;

    return NPERR_NO_ERROR;
  }
    else
      return NPERR_OUT_OF_MEMORY_ERROR;
                                     // Couldn't get enough memory
}
```

NPP_Destroy

```
// NPP_Destroy: ( from npshell.cp ) This procedure destroys a PluginInstance.  First
// the data pointer's memory is freed, then the controls are destroyed, and finally
// the instance itself is freed and set to zero (so that it won't be errantly used again).

NPError NPP_Destroy(NPP instance, NPSavedData** save)
{
  if (instance == NULL)              // Check for invalid plug-in instance
    return NPERR_INVALID_INSTANCE_ERROR;

  Plug-inInstance* This = (Plug-inInstance*) instance->pdata;

  if (This != NULL)
  {
    if (This->data)
      NPN_MemFree(This->data);       // Free up data pointer

    if ( This->decControl != NULL ) // Destroy the controls
      DisposeControl( This->decControl );
    if ( This->incControl != NULL )
      DisposeControl( This->incControl );

    NPN_MemFree(instance->pdata);    // Free instance data
    instance->pdata = NULL;          // Set instance to NULL
  }

  return NPERR_NO_ERROR;
}
```

NPP_SetWindow

```
// NPP_SetWindow: ( from npshell.cp ) This procedure sets our
// PluginInstance's window to the window passed as a parameter
// in this procedure.

NPError NPP_SetWindow(NPP instance, NPWindow* window)
{
  if (instance == NULL)
    return NPERR_INVALID_INSTANCE_ERROR;

  Plug-inInstance* This = (Plug-inInstance*) instance->pdata;

  This->fWindow = window;          // Set this instance window to the NPwindow

  return NPERR_NO_ERROR;
}
```

NPP_NewStream

```
// NPP_NewStream: (from npshell.cp) Sets the amBusy boolan to true, indicating that
// we are ready to load data from the .test file.

NPError NPP_NewStream(NPP instance,
                NPMIMEType type,
                NPStream *stream,
                NPBool seekable,
                uint16 *stype)
{
  if (instance == NULL)
    return NPERR_INVALID_INSTANCE_ERROR;
  Plug-inInstance* This = (Plug-inInstance*) instance->pdata;

  This->amBusy = TRUE;             // Loading data...

  return NPERR_NO_ERROR;
}
```

NPP_WriteReady

```
// NPP_WriteReady: ( from npshell.cp ) Returns the value of our read buffer.
// (Navigator will be doing the writing, our plug-in will be reading.)

int32 NPP_WriteReady(NPP instance, NPStream *stream)
{
  return kBufferSize;              // Return our buffer size
}
```

NPP_Write

```
// NPP_Write: ( from npshell.cp ) Loads the data from the .test file into the
// PluginInstance.

int32 NPP_Write(NPP instance, NPStream *stream, int32 offset,
                  int32 len, void *buffer)
{
  if (instance == NULL)            // Check for invalid plug-in instance
    return NPERR_INVALID_INSTANCE_ERROR;

  Plug-inInstance* This = (Plug-inInstance*) instance->pdata;

  if (This != NULL)
    GetData(This, len, buffer);    // Load the .test file's data into
                                   // This->data
  return 0;
}
```

NPP_DestroyStream

```
// NPP_DestroyStream:  ( from npshell.cp ) When finished reading data, the plug-in
will
// set amBusy to false, set the min, max and current values of our PluginInstance,
// draw the controls, and draw the indicator.

NPError NPP_DestroyStream(NPP instance, NPStream *stream,
                          NPError reason)
{
  if (instance == NULL)            // Check for invalid plug-in instance
    return NPERR_INVALID_INSTANCE_ERROR;

  Plug-inInstance* This = (Plug-inInstance*) instance->pdata;

  This->amBusy = FALSE;            // Finished loading...

  if (SavePort(This->fWindow))     // Save the current GrafPort
  {
    SetValues( This );             // Set min, max and current
    AddControls( This );           // Draw the controls
    DrawContents(This);            // Draw the indicator
    RestorePort(This->fWindow);    // Restore the old GrafPort
  }

  return NPERR_NO_ERROR;
}
```

NPP_StreamAsFile

```
// NPP_StreamAsFile: ( from npshell.cp )

void NPP_StreamAsFile(NPP instance, NPStream *stream, const
                      char* fname)
{
// QuickTest doesn't support loading files
}
```

NPP_HandleEvent

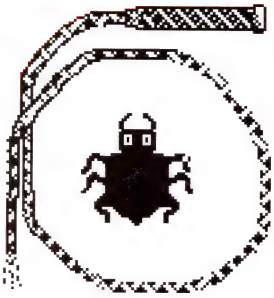
```
// NPP_HandleEvent: ( from npshell.cp ) Takes a pointer to an event, casts it to
// an EventRecord* and handles the event. This plug-in only handles update and
// mousedown events.

int16 NPP_HandleEvent(NPP instance, void* event)
{
  Boolean eventHandled = false;// Has the event been processed?
  WindowPtr theWindow;         // Window where mouse was pressed
  short wherePressed;          // Coordinates where mouse was pressed
  EventRecord*ev;              // Event record for this event
  NP_Port  *port;              // The window for mouse click

  if (instance == NULL)        // Check for invalid plug-in instance
    return eventHandled;

  Plug-inInstance* This = (Plug-inInstance*) instance->pdata;
  if (This != NULL && event != NULL)
  {
    ev = (EventRecord*) event; // Convert the event to an EventRecord
    switch (ev->what)          // Get the type of event
    {
      case updateEvt:          // Update event:
        if( SavePort( This->fWindow ) )      // Save the current
                                             // GrafPort
        {
          DrawContents(This);                // Draw the contents
          UpdateCntrls( This->fWindow );     // Update the controls
          RestorePort( This->fWindow );      // Restore the old GrafPort
        }
        eventHandled = true;   // Event was processed
        break;
      case mouseDown:          // Mouse down:
        port = (NP_Port*) This->fWindow->window;
                               // Get the GrafPort from the
        theWindow = ( WindowPtr ) port->port;
                               // fWindow data structure
        if ( theWindow != FrontWindow() )
                               // If window isn't front window
          BringToFront( theWindow );
                               // bring it to the front.
            else
            {
              wherePressed = FindWindow( ev->where,
                                         &theWindow );
                                     // Find where mouse down
              switch ( wherePressed )  // If mouse press down in...
              {
                case inContent:      // ...content...
                  if( SavePort( This->fWindow ) )
                                     // Save current GrafPort
                  {
                    HandleContents( This, ev->where,
                                    theWindow );
                                     // Handle the mouse down event
                    RestorePort( This->fWindow );
                                     // Restore the old GrafPort
                  }
                  break;
              }
            }
        break;

      default:
        break;
    }

  }
    return eventHandled;// Let Navigator know if we processed the event
}
```

Gives you the Information to *Program your Best!*

The Debugger V2 & MacNosy

by Steve Jasik

Information



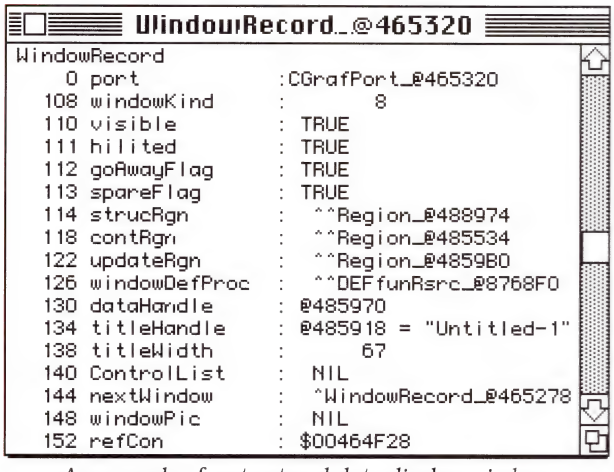
Control

For over 10 years Steve Jasik has brought Macintosh Developers the superior debugger. In addition to the software, Steve provides timely support to his customers. Inspect the feature list and see why so many developers use it. In addition to basic features such as stepping and breakpoints, the Jasik debugger has many advanced features such as heap display, a fast memory watch, Soft-MMU, jump tracing, ... to help you track down the elusive bugs that other debuggers won't help you find.

In additon, the package includes **MacNosy,** a global interactive disassembler that enables one to recover the source code of any Mac application or from the ROM.

When you compare features of the different debuggers, note that *only one* has all the below features to help you get your job done, and *only one* has MacNosy to help you debug any program in a full system environment symbolically!

"The Debugger" is the debugger of choice at: Adobe, Apple, Claris, Kodak, Metrowerks, etc.



An example of a structured data display window with hypertext links to substructures

Its Features Include:

- **Source level debugging for Metrowerks & MPW** compiled programs (C++, C, Pascal, Fortran, ...).
- **Symbolic Debugging** of *any* Macintosh program, ROM, Plug-In or *code resource* (DRVRs, XCMDs, INITs, PDEFs..), ASLM & **CFM LIbraries, OpenDoc parts**, GX Drivers, ...
- **Simultaneous Symbolic debugging of multiple "tasks"**
- **Object Inspector for MacApp 3 & PowerPlant programs**
- **Source level debugging of Symantec C™ projects**
- **Includes a program (CoverTest) to interactively do Code Coverage analysis for SQA testing, etc.**
- **Fast Software Watchpoint** to find clobbered variables.
- **Soft MMU** ("heap centric" bounds checking) for PowerPC programs to find out of bounds references.
- **Jump Trace** to find where a program takes a "wild jump"
- **Sophisticated error check algorithms** such as Trap Discipline (Argument Checking), Heap Scramble and Heap Check to detect errors before they become disasters
- **Structured display** of data (hypertext) with user definable structures while debugging
- **Conditional breakpoints** to filter redundant information
- **Continuous Animated Step Mode** to watch your program execute instruction by instruction
- **Detailed symbolic disassembly for both 680x0 and PowerPC** with symbol names, labels, cross ref maps, – make it possible to ferret out the secrets of the ROM, etc.
- **"Training Wheels"** for the PowerPC disassembler to help you learn the opcodes

The Debugger V2 & MacNosy: $350

Runs on all Macs. Call For Group prices or Updates. Visa/MC Accepted.

Available from: Jasik, APDA or ComputerWare (800-326-0092).

Jasik Designs • 343 Trenton Way, Menlo Park, California 94025 • (415) 322-1386

URL: http://www.jasik.com e-mail: macnosy@jasik.com

NPP_Print

```
// NPP_Print: ( from npshell.cp ) For brevity this procedure isn't implemented.
// The code in this procedure does the minimal processing required to make printing
// work (note that the plug-in's indicator and controls are not printed - see the
// examples in the Plug-in SDK for information on printing within plug-ins).

void NPP_Print(NPP instance, NPPrint* printInfo)
{
  if (instance != NULL)
  {
    if (printInfo->mode == NP_FULL)      // in fullscreen mode we don't do
                                         // anything for printing.
      printInfo->print.fullPrint.plug-inPrinted = false;
  }
}
```

NPP_URLNotify

```
// NPP_URLNotify: Not implemented

void NPP_URLNotify(NPP instance, const char* url,NPReason
reason, void* notifyData)
{
// QuickTest doesn't implement this method
}
```

NPP_GetJavaClass

```
// NPP_GetJavaClass: Not Implemented (NOTE: this gives a warning during Make)

jref NPP_GetJavaClass(void)
{
// QuickTest doesn't implement this method
}
```

SavePort

```
// SavePort: Since Mac plug-ins share the drawing environment with Navigator we
// MUST save the current GrafPort settings. This subroutine saves the current port's
// clipping rectangle. Save other port settings before you change them.

Boolean SavePort(NPWindow *window)
{
  Rect clipRect;
  NP_Port* port;
  if (window == NULL)
    return FALSE;

  port = (NP_Port*) window->window;
  if (window->clipRect.left < window->clipRect.right)
  {
  // Preserve the old port
    ::GetPort((GrafPtr*)&gOldPort);
    ::SetPort((GrafPtr)port->port);

  // Preserve the old drawing environment
    gSavePort.portRect = port->port->portRect;
    ::GetClip(gSavePort.clipRgn);

  // Setup our drawing environment

    clipRect.top = window->clipRect.top + port->porty;
    clipRect.left = window->clipRect.left + port->portx;
    clipRect.bottom = window->clipRect.bottom + port->porty;
    clipRect.right = window->clipRect.right + port->portx;
    ::SetOrigin(port->portx,port->porty);
    ::ClipRect(&clipRect);
    clipRect.top = clipRect.left = 0;
    return TRUE;
  }
  else
    return FALSE;
}
```

RestorePort

```
// RestorePort: restore the old port settings so Navigator has the same GrafPort settings

void RestorePort(NPWindow *window)
{
  NP_Port* port;
  CGrafPtr myPort;

  port = (NP_Port*) window->window;
  ::SetOrigin(gSavePort.portRect.left,
gSavePort.portRect.top);
  ::SetClip(gSavePort.clipRgn);

  ::GetPort((GrafPtr*)&myPort);
  ::SetPort((GrafPtr)gOldPort);
}
```

DrawContents

```
// DrawContents: draw the indicator. Indicator is made up of on and off Picts which
// must be loaded from the plug-in's resource fork.

void DrawContents(Plug-inInstance *This)
{
  Rect drawRect;                 // Drawing rectangle
  short  loop;                   // Loop to process indicator

    drawRect.top = 0;            // Set the initial draw rectangle
    drawRect.bottom = 34;
    drawRect.left = 0;
    drawRect.right = 17;

    for ( loop = This->min; loop <= This->current; loop++ )
                                 // Draw the ON picts for
    {                            // the indicator
      ::DrawPicture( (PicHandle) gonPict, &drawRect );
      drawRect.left += 23;       // Move the draw rectangle
      drawRect.right = drawRect.left + 17;
                                 // to the right
    }

    for ( loop = This->current+1; loop <= This->max; loop++ )
                                 // Draw OFF picts for the indicator
      ::DrawPicture( (PicHandle) goffPict, &drawRect );
      drawRect.left += 23;       // Move the draw rectangle
      drawRect.right = drawRect.left + 17;
                                 // to the right
    }

}
```

HandleContents

```
// HandleContents: process mouse clicks in the contents of the window. Track clicks
// in the controls and process those clicks.

void HandleContents( Plug-inInstance *This, Point where,
                     WindowPtr theWindow )
{
  int part, thePart;             // part mouse down occurred
  ControlHandle theControl;      // Control mouse was pressed in
  Str255 theTitle;               // Control's title

  GlobalToLocal( &where );       // Convert coordinates
  part = FindControl( where, theWindow, &theControl );
                                 // Find control were mouse
                                 // down occurred
  if ( theControl != NULL )      // If control valid
  {
    thePart = TrackControl( theControl, where, nil );
                                 // Track the press
    if ( thePart == inButton ) // If release in button
    {
      ::GetControlTitle( theControl, theTitle );
                                 // Get the title of the control
      if ( EqualString( theTitle, kDecrease, false, false ) )
      {                          // If decrease title...
        This->current--;         // Decrease indicator's current value
        if ( This->current < This->min )
                                 // Make sure current >= min
        {
```

```
            This->current = This->min;      // Make current = min
              ::SysBeep( 10 );
            }
            DrawContents( This );           // Draw the contents
          }
          if ( EqualString( theTitle, kIncrease, false, false ) )
          {                                 // If increase title...
            This->current++;                // Increase indicator's current value
            if ( This->current > This->max )
                                            // Make sure current <= max
            {
              This->current = This->max; // Make current = max
              ::SysBeep(10);
            }
            DrawContents( This );           // Draw the contents
          }
        }
      }
    }
```

UpdateCntrls

```
// UpdateCntrls: updates the controls for the window.

void UpdateCntrls( NPWindow *window )
{
  WindowPtr   theWindow; // Window pointer for window which contains controls
  NP_Port     *port;     // NP_port which points to GrafPort

  port = ( NP_Port* ) window->window;    // Convert fWindow
  theWindow = ( WindowPtr ) port->port; // to a WindowPtr

  UpdateControls( theWindow, theWindow->visRgn );
                          // Update the window's controls
}
```

GetData

```
// GetData: Get the data out of the buffer and put it into the data pointer of
// our plug-in instance.

void GetData( Plug-inInstance *This, unsigned long len, void
*buffer )
{
  char *newText;
  long offset;

  if (This->data == NULL)                       // No data loaded yet
  {
    newText = (char*) NPN_MemAlloc(len);  // Allocate newText
    This->datalength = 0;                 // Set length of data
    offset = 0;                           // Set initial offset
  }
  else                                          // We've loaded data
  {
    newText = (char*) NPN_MemAlloc(This->datalength+len);
    BlockMove(This->data, newText, This->datalength);
    NPN_MemFree(This->data);
    offset = This->datalength;
  }
  BlockMove(buffer, newText+This->datalength, len);
                                                // Move buffer data
  This->data = newText;                         // info data
  This->datalength += len;                      // and set the length
}
```

SetValues

```
// SetValues: convert data to min, max and current values.  If values are out
// of bounds then set min, max and current to default values.

void SetValues( Plug-inInstance *This )
{
  This->min = GetValue( This->data );
                                // Set min
  This->max = GetValue( ( This->data ) + 7 );
                                // Set max
  This->current = GetValue( ( This-> data ) + 14 );
                                // set current

  if ( ( This->min > This->current ) || ( This->max <
          This->current ) )
    SetDefaults( This );        // Use default if current too low or too high
```

```
  if ( ( This->min < kMin ) || ( This->min > kMax ) )
    SetDefaults( This );      // Use default if min too low or too high

  if ( ( This->max < kMin ) || ( This->max > kMax ) )
    SetDefaults( This );      // Use default if max too low or too high

  if ( This->max < This->min )
    SetDefaults( This );      // Use default if min greater than max
}
```

SetDefaults

```
// SetDefaults: If min, max or current out of range set all three to defaults

void SetDefaults( Plug-inInstance *This )
{
  This->min = kMin;
  This->max = kMax;
  This->current = kMin;
}
```

GetValue

```
// GetValues: read three bytes starting at bufferstart+4 ( to skip over the
// min=, max=, or cur= ) and convert it to an integer.

int GetValue( Ptr bufferstart )
{
  int value = 0;

  bufferstart = bufferstart + 4;
  value = ( ( *(bufferstart) - 48 ) * 100 ) +
    ( ( *(bufferstart + 1 ) - 48 ) * 10 ) +
    ( *(bufferstart + 2 ) - 48 );
  return value;
}
```

AddControls

```
// AddControls: add the controls for increase and decrease to the window

void AddControls( Plug-inInstance *This )
{
  Rect   drawRect;
  WindowPtr  theWindow;
  NP_Port    *port;

  port = (NP_Port*) This->fWindow->window; // Convert the fWindow
  theWindow = ( WindowPtr ) port->port;    // to a window pointer

  SetRect( &drawRect, 10, 46, 100, 71 );
                                  // Rect for the decrement control
  This->decControl = NewControl( theWindow, &drawRect,
              "\pDecrease", true, 0, 0, 1, pushButProc, 0);

  SetRect( &drawRect, 120, 46, 210, 71 );
                                  // Rect for the increment control
  This->incControl = NewControl( theWindow, &drawRect,
              "\pIncrease", true, 0, 0, 1, pushButProc, 1);
}
```

Add QuickTest.cp to the QuickTest project. Change the project's preferences File name to QuickTest68K (or QuickTestPPC if you are creating a PPC plug-in) and the SYM name to QuickTest68K.SYM if you are implementing a 68K plug-in (see figure 3) . Finally, you need to add the Plug-in SDK's Include folder to the access path. You can do this by either adding the Include folder to the access path with the Access Path option of the Preferences dialog box or copying the Include folder into the Quick Test folder.

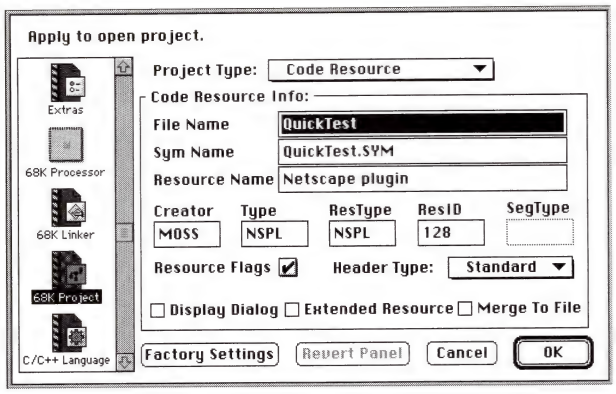


Figure 3: Screen shot of QuickTest project preferences

Select Make to compile the project. When the project successfully compiles, drop the resulting binary into the Plug-ins folder, which is in the same folder as the Navigator application, and restart Navigator if it is running.

Next, we need to create an HTML test document and a document, with a name that ends in the extension .test, that holds our data for the indicator.

Here's the HTML:

```
<TITLE>Plug-in Test</TITLE>
Here are the indicator and controls:<BR>
<EMBED SRC="example1.test" ALIGN=CENTER WIDTH=470
HEIGHT=75><P>
This is a quick sample plug-in.<P>
```

and the .test document referenced in the above HTML:

```
min=001max=015cur=005
```

Now we can test the plug-in. Drag the test HTML file's icon to Navigator's icon. The indicator will be displayed along with the controls to change the indicator's value. Try changing the HTML and .test file (you can even have multiple occurrences of the plug-in within one HTML document) to see how the HTML and .test files interact with the plug-in.

So, how exactly does QuickTest work? First, you'll notice that there isn't a main method. Instead there are several methods with names that begin with NPP and several other methods that are called by those NPP methods. Navigator will be calling these your plug-in's NPP routines, controlling how your plug-in gets executed.

First, Navigator calls the plug-in's NPP_Initialize method to set the clipping region of the plug-in's window, and loads the resources from QuickTest's resource fork for the on and off PICTs of the indicator. NPP_Initialize is only called once, the first time a plug-in is loaded.

Next, Navigator calls QuickTest's NPP_New method to initialize an instance of a plug-in, setting the instance's variables to their initial values. NPP_SetWindow is then called to set a window for the newly created instance.

To read in the data file which ends with the .test (in this example example1.test), Navigator calls NPP_NewStream, which sets the instance's amBusy boolean to true. This boolean indicates that the instance is reading data. NPP_WriteReady is then called to determine the number of bytes that will be read in at one time; in QuickTest's case this is one byte. Navigator then calls NPP_Write to write out bytes from the .test file to the plug-in. NPP_Write is continually called until all the data is read from the file. QuickTest reads this data in and sets the instance's data pointer to point to what was read. After the data is read Navigator calls NPP_DestroyStream which sets amBusy to false.

At this point in NPP_DestroyStream we save the current GrafPort settings, convert the data pointed to by data to minimum, maximum , and current values with SetValues, add the controls to the instance's window, and finally, draw the indicator with DrawContents. After this the GrafPort settings are restored.

The user then can click on the controls to change the indicator's setting. Any clicks, keystrokes, etc. cause Navigator to call QuickTest's NPP_HandleEvent method. This method checks which control is pressed and increments or decrements the indicator. NPP_HandleEvent also handles update events for redrawing the screen when an instance's window needs to be updated.

When the user goes to another page the instance is destroyed with NPP_Destroy. In QuickTest, NPP_Destroy frees the memory used by the instance and then destroys the instance by setting the instance to NULL. This is also a precautionary measure, since all NPP routines in QuickTest check for a NULL instance before doing anything with the instance. When the plug-in is unloaded, NPP_Shutdown is called to dispose of the clip region and release the PICT resources.

Adding a better input checking feature and displaying a value for the minimum and maximum would be two nice enhancements for this plug-in.

Conclusions

If you write you own plug-in which uses a MIME type that isn't already registered, you should register the MIME type so that the MIME type will be reserved for your plug-in. Information on registering MIME types is available at:

http://home.netscape.com/assist/helper_apps/rfc3.html

The future for plug-ins looks encouraging. Microsoft's Internet Explorer 2.0 also supports the same plug-in structure as Navigator, and Netscape's latest SDK (version 3.0, currently in beta) includes documentation to integrate plug-ins with Java and JavaScript. This means that a majority of users surfing the web have the capability to use plug-ins that you produce. Given the speed and flexibility of plug-ins, many software companies are rushing to produce plug-ins for their document types, including Adobe's Acrobat documents and RealAudio's RealAudio audio files.

MT

By Lloyd Chambers, Senior Manager/Software Development,

Simple Yet Effective Bug Detection

Detecting and preventing common programming errors

Introduction

Most programmers are familiar with polling versus notification. Polling is when you continually scan for events to process until you find something of interest; notification is when you are told that there is something important to attend to, and unless there is something to attend to, you can use the time for something else. Polling is inefficient and wastes resources; notification is a far superior approach because you only expend effort as needed.

While programmers agree that polling is undesirable, few of them would be willing to admit that's what they do everyday when they debug code—they poll for bugs until they find one by running the debugger, reading code, etc. This article discusses how to eliminate the old, inefficient polling paradigm of finding bugs with a new paradigm of bug notification.

This new bug-notification paradigm doesn't apply to all types of bugs, but it does apply to a certain class of bugs that can be systematically eliminated while imposing little or no additional burden on the programmer. The three types of programming errors commonly responsible for a large number of bugs are: (1) invalid routine parameters, (2) failure to detect errors, and (3) memory leaks.

This article is divided into four major areas of interest:

- how to set up a debugging system;
- how to create and use asserts;
- how to create debugging versions of system calls;
- how to detect memory leaks.

You will want to read this article from start to finish as later techniques build on earlier ones.

This article avoids discussing techniques that require significant changes in coding style, extra work on the part of programmer or more than beginner-level concepts to use properly. Other techniques can be useful but offer limited "bang for the buck," especially for beginning programmers. The techniques discussed here will be invaluable to all programmers regardless of experience.

The techniques discussed in this article could apply to any environment, though the implementations shown are for C and C++ environments. The provided sample code should be easy to retrofit into existing C or C++ projects. Complete implementations are not provided with this article, but enough functionality is provided to be quite useful "as is."

Setting up a debugging system

Before moving into specific debugging techniques, we need to discuss how debugging code in general should be implemented. There are two general requirements debugging code must meet—it must be able to detect and report programming errors, and it must be easy to remove it from the product without altering source code.

Debugging code must be able to report programming errors. A reporting system can be simple or elaborate. At Symantec, we have chosen a simple system that is suitable for use in any type of code whether it be an application, INIT, driver, etc. More elaborate reporting systems can be devised,

Lloyd Chambers has programmed for fifteen years. He is the designer and primary author of DiskDoubler and AutoDoubler and holds several compression related patents. Lloyd co-founded Salient Software, Inc, which was acquired by Fifth Generation Systems, Inc, which was later acquired by Symantec. Lloyd can be reached electronically at lloyd@llc4.com.

but they may then become limited in their applicability to certain types of code and may themselves interfere with the ability to debug code. Consequently, our basic reporting mechanism is a debugger break. Though it is primitive, it does the job, and it is simple, non-intrusive, and applicable to all types of code. Note that a high-level debugger which allows you to step through code is not a system to detect bugs— rather it is a tool you use to poll for bugs.

The second requirement of a bug detection system is that it can be removed from the final product so that it causes no run-time overhead. This leads to the building of a "debug" version and a "non-debug" version. The debug version may run more slowly and will be larger due to the additional debugging code; however, the debugging code disappears when a non-debug version is produced. When these two requirements are met, the programmer can detect bugs, yet the end-user realizes full performance from the product. *It is important to realize that the mechanisms discussed here are focused on detecting programmer errors, not runtime errors that can reasonably be expected to occur.* As such, all the mechanisms discussed are appropriate for a debug, testing version targeted at helping the programmer eliminate bugs from code. The mechanisms discussed here are not intended as an error handling mechanism for your program; those sorts of things must be handled using other techniques, and ultimately may involve reporting problems to the end-user.

The DEBUG flag

Compiling a debug or non-debug program is best done using a compiler flag. We define a flag called `DEBUG`. In most programming environments, you can define such a flag in a prefix file which will be included for all source files. With multiple projects, you can have each project prefix file include a shared file which defines the `DEBUG` flag. In this way, you can exercise global control at build time over whether a debug or non-debug version is built, even if your product consists of dozens of modules. The `DEBUG` flag should be defined as 0 for debugging to be off, and 1 if it should be on:

```
#define DEBUG 1 // debugging on
```

All debugging code not intended for the final project should be dependent upon this flag. Debugging macros should be defined to go away when `DEBUG` is off. Code that is debugging only should be conditionally compiled. For example:

```
#if DEBUG
    #define DebugMsg( msg )    { DebugStr( msg ); }
#else
    #define DebugMsg( msg )    {/*nothing*/}
#endif

#if DEBUG
  // debugging code here
#endif
```

In addition, other more specific compiler flags may be dependent on the `DEBUG` flag. In general, if you have a specific debugging module, you'll want to use a flag for it which is dependent on the `DEBUG` flag. This is useful to selectively enable or disable certain debugging capabilities without disabling other debugging capabilities. By default, it could be the same as the `DEBUG` flag, but could also be set differently. For example:

```
#if DEBUG
   #define USE_DEBUG_TRAPS   1     // on or off as desired
#else
   #define USE_DEBUG_TRAPS   0     // always off when DEBUG off
#endif
```

Creating and using Asserts

Now that we've discussed how to set up a debugging system, let's move on to the basic building blocks most debugging code will use. Verifying program requirements is done using "assert" calls and related variants. Asserts allow the programmer to require and flag run-time conditions that do not meet program requirements. For example, a routine that takes a pointer may want to require that it be non-nil. Asserts are used in a number of programming environments and will differ from the implementation discussed here.

Asserts may be very simple or quite complex. They are best implemented as macros so that they can be compiled out of the code in a non-debug version. Note that a macro may call a routine, or may use inline code, depending on its complexity. Here are the asserts and related variants we commonly use at Symantec:

```
// break with message if condition is false (zero)
Assert( condition, failureMessage )

// break with message if false; append number to it
AssertNum( condition, failureMessage, theNumber )

// break into debugger with message
DebugMsg( message )

// break with message if true (non-zero)
DebugMsgIf( condition, message )

// break with message; append number to it
DebugNum( message, theNumber )

// break with message if true; append number to it
DebugNumIf( condition, message, theNumber )
```

For example, if you want to verify that a pointer is non-nil, you would write the following assert:

```
Assert( thePointer != nil,
  "\pRoutineName: nil pointer");
```

You can also use higher level asserts which perform a fair amount of computation to verify the assertion. In general, more complex asserts verify the integrity of a program entity. The following asserts break into the debugger with the specified message *plus* additional useful information when there is a problem:

```
// break if the Handle is invalid
AssertHandleIsValid( theHandle, failureMessage )

// break if the Handle is invalid or not a resource
AssertResourceIsValid( theResource, failureMessage )

// break if the address is invalid
// or not aligned to the specified boundary
AssertAddressIsValid( addr, failureMessage )
```

```
AssertAddressIsValidAlign( addr, alignBy, failureMsg )
AssertAddressIsValidAlign2( addr, failureMsg )
AssertAddressIsValidAlign4( addr, failureMsg )

// break if 'fsSpec' is not a valid FSSpec
// also displays FSSpec and describes why it is invalid
AssertSpecIsValid( fsSpec, failureMsg )

// break if 'refNum' is not a valid ref num
AssertFileRefNumIsValid( refNum, failureMsg )

// break if 'upp' is not a valid universal proc ptr (PPC)
AssertUPPIsValid( upp, failureMsg )

// break if 'string' is too long or too short
AssertStringIsValid( string,
  minLength, maxLength, failureMessage )

// break if 'desc' is invalid
AssertAEDescIsValid( desc, failureMsg )

// break if err is anything other than 'noErr' or a cancel
// also displays the error code
AssertNoErr( err, failureMsg );
DebugIfErr( err, failureMsg );
```

How to declare Asserts and other debugging routines

Debug code should be defined to compile in when `DEBUG` is on, and compile to nothing when `DEBUG` is off. In the following example, the thing to notice is that when `DEBUG` is off, the macro is defined in such a way that no code is generated by the compiler:

```
#if DEBUG
  void _AssertHandleIsValid( const void *theHandle,
          const char  *varName, ConstString255Param msg );

  #define AssertHandleIsValid( h, msg ) \
      _AssertHandleIsValid( h, #h, msg );
#else
  #define AssertHandleIsValid(h, msg )  {/*nothing*/}
#endif
```

You could also choose to define the routine as a normal function in a debug version and a macro that does nothing in a non-debug version. However, that approach is less flexible. For example, as a macro, `AssertHandleIsValid` calls `_AssertHandleIsValid` with an additional parameter generated by the preprocessor '#' operator. Suppose your code looks like this:

```
AssertHandleIsValid( myThing->itemList, "\pMyRoutine" );
```

The code actually seen by the compiler is as follows:

```
_AssertHandleIsValid( itemList,
    "myThing->itemList", "\pMyRoutine" );
```

The parameter `varName` is a C string which is the textual version of the parameter or expression. When `AssertHandleIsValid` detects an invalid Handle, it generates an error message describing the problem and incorporates this string into the error message. Note that some older development environments may not support the `#x` preprocessor directive.

An example Assert

`_AssertHandleIsValid` checks as many things as possible to verify the validity of the Handle. When a Handle can

make it through this routine, it is highly probable that it is valid. If a problem is found, a detailed message describing the problem is generated and a debugger break is made.

```
  void
_AssertHandleIsValid(
  const void *    theHandle,
  // void * avoids need to cast
  const char *    varName,
  // a null terminated C string
  ConstStr255Param  msg)
  {
  OSErr  err;
  Str255 result;

  result[0]  = 0;

  if ( IsNil( msg ) )
    {
    msg  = "\pAssertHandleIsValid()";
    }

  _AssertAddressIsValidAlign( theHandle, varName, 4, msg);

  /*
  make a memory reference to force a crash here if the handle is invalid. It is better to
  force a crash here, rather than crashing in some obscure place in the ROM later

  call a dummy routine to hopefully prevent compiler from optimizing out our
  dereference.
  */

  MemoryDummy( *(short *)theHandle );

  (void)HandleZone( (Handle) theHandle );
  err  = MemError();
  if ( IsErr( err ) )
    {
    Str255 errStr;

    BuildBadAddressMsg( result,
      "\perror from HandleZone:", varName, msg);

    DebugGetErrorString( err, errStr );
    AppendPString( "\p: ", result);
    AppendPString( errStr, result);

    DebugMsg( result );
    }

  if ( IsntNil( *(Handle)theHandle ) )
    // if it has a master pointer, then
    // check to see if we can get its size
    {
    (void)GetHandleSize( (Handle) theHandle );
    err  = MemError();
    if ( IsErr( err ) )
      {
      Str255 errStr;

      BuildBadAddressMsg( result,
        "\perror from GetHandleSize:", varName, msg);

      DebugGetErrorString( err, errStr );
      AppendPString( "\p: ", result);
      AppendPString( errStr, result);

      DebugMsg( result );
      }
    }
  }
```

The following sample code generates the subsequent debugger breaks:

```
  Handle theHandle  = nil;
  AssertHandleIsValid( theHandle, "\pmain");
```

```
User break at 041B6268 _AssertAddressIsValidAlign+000AA
  NIL address: 'theHandle' [main]
User break at 041B637E _AssertHandleIsValid+000E6
 error from GetHandleSize: 'theHandle' [main]: nilHandleErr
```

Registering error codes

Some asserts display error codes. In the `_AssertHandleIsValid` example, shown above, `DebugGetErrorString()` is called to get a string for the error. Rather than display a raw number, such as -109, it's often more convenient to see a string instead such as `nilHandleErr`. This is particularly nice when error codes change between revisions. And for less technical people, the message may be more easily remembered than a numeric error code. The supplied sample code implements this idea; see `AddDefaultErrorStringTables()` in Debug.c. You can also add your own error codes by calling `DebugAddOSErrStringTable()`.

Using Asserts and their variants

We've now discussed how to create asserts. This next section discusses when to use them.

A common and critical place to use asserts is at the beginning of a program routines. All routines should be written using asserts to notify the programmer of illegal or questionable parameters. Depending on your overall design, your routines may or may not choose to check for illegal parameters in a non-debug version, but in a debug version there is no good reason for not detecting a client calling error. You will save your client (often yourself) much debugging effort if you fail an assert when an invalid call is made. In addition, asserts can often take the place of certain types of comments, which can become repetitive and out of date. Asserts cannot be out of date, since they are executable code that will complain if their requirements are not met.

It is important to remember that the primary purpose of asserts is to flag programming errors, not to handle problems that can arise naturally during programming execution. Handling of normal runtime errors is best handled through an error handling scheme, which of course is part of the final product. Asserts can be used to flag such conditions, but do not take the place of an error-handling scheme which prevents the program from crashing. You will probably want to use asserts *and* an error-handling scheme. If you use an error-handling scheme that silently cleans up without notification, you are not benefitting from bug detection. Also, it is unlikely that an error handling scheme will check anywhere near as stringently as an assert might. For example, your routine might reasonably be expected to check for a nil Handle, but only a `DEBUG` version would reasonably be expected to verify the integrity of the Handle as does `AssertHandleIsValid`.

The following sample routine documents its parameters well using asserts. If there are additional requirements for the parameters, they too should be documented using Asserts, particularly any non-obvious requirements. Note that the routine also flags any resulting error code. This may or may not be

desirable, depending on whether you want to know about the error at this point. Often, flagging an error is a great idea, because the caller may not be checking the error result.

```
  OSErr
DoSomething
  (
  Handle              theHandle,
  ConstString255Param fileName,
  void *              buffer,
  UniversalProcPtr    callback,
  short               fileRefNum
  )
  {
  OSErr  err  = noErr;

  AssertHandleIsValid( theHandle, "\pDoSomething");
  AssertStringIsValid( fileName, 1, 31, "\pDoSomething");
  AssertAddressIsValid( buffer, "\pDoSomething");
  AssertUPPIsValid( callback, "\pDoSomething");
  AssertFileRefNumIsValid( fileRefNum, "\pDoSomething");

  ... useful code goes here ...

  err = SomethingElse();

  AssertNoErr( err, "\pDoSomething");
  return( err );
  }
```

Consistent use of the above technique with every routine you write will provide major benefits, especially on team projects, where not everyone is familiar with the code. Even if it's only yourself, you will be grateful for the notification when you call it incorrectly one day. In some cases, when you review your code, the asserts are the only clue you'll have as to the proper value of the parameters. In such cases, they can save you much time and aggravation when trying to determine what the programmer intended. They also demonstrate whether the programmer thought about what requirements the routine had when writing it. When asserts are missing, it is often unclear what requirements are present unless you carefully read through the code—an onerous task in some cases. Those familiar with the theoretical aspects of program correctness will recognize that a prerequisite of proving code correct is establishing its calling requirements. Asserts document those requirements.

It should be noted that within a routine, certain areas of the code may have their own assumptions and requirements. These are good places to add asserts which document those conditions at that point in the code.

Note that although asserts do increase source-code size, they often can take the place of comments which invariably become inaccurate as the code is modified. They are "smart comments" because they make explicit the requirements, and guarantee the requirements are met when the routine is called. Such an approach is vastly superior to a comment that says "must not be nil" because an Assert is actually enforced at runtime (at least in the DEBUG version). Forget those types of comments and use Asserts instead.

Debugging "Traps"

We've now discussed creating and using asserts. This next section discusses how to create debugging versions of system calls using asserts and other techniques.

For many years I've wished that Apple would provide a debugging version of the ROM. Years went by before I thought of the following approach which effectively lets you write debugging versions of system calls such as `NewHandle`. There have been various tools that have made such checking possible, but they all have limitations of various sorts. Use of the following technique requires no change to your source code and can readily be applied to older source code bases. It could also be applied to non-system routines or other library routines.

The technique is simple: use the C/C++ macro preprocessor to substitute your own debugging version of a call for the system one. Your version makes appropriate debugging checks, calls the real routine, and if appropriate, checks the results of the call. With such an approach you can verify the validity of parameters and flag any error conditions that result from the call. Your checks can be very stringent. For example, suppose you want to exercise debugging code whenever `DisposeHandle()` is called. Symantec's implementation performs the following checks:

- verifies that the Handle is valid;
- verifies that it is not a resource;
- sets all bytes of the Handle to a garbage value;
- verifies that no error occurred.

```
  pascal void
DebugTraps_DisposeHandle(Handle h)
  {
  UInt32 hSize;
  SInt8    hState;

  AssertHandleIsValid( h, "\pDebugTraps_DisposeHandle");

  hState = HGetState( h );
  Assert( ! HStateIsResource( hState ),
    "\pDebugTraps_DisposeHandle: This Handle is a resource.
     Use ReleaseResource instead");

  // fill the handle with garbage prior to disposing it
  hSize  = GetHandleSize( h );
  if ( MemError() == noErr )
    {
  // note: FillWithGarbage doesn't move memory so
  // Handle doesn't need to be locked
    FillWithGarbage( *h, hSize);
    }

  DisposeHandle( h );

  // use LMGetMemErr instead of MemError so
  // error code doesn't get cleared
  AssertNoErr( LMGetMemErr(),
       "\pDebugTraps_DisposeHandle");
  }
```

Setting the contents of the Handle to garbage values has the major benefit of flushing out other bugs. Any other code that retains a reference to a disposed Handle should crash or start acting strange soon thereafter, rather than much later. Whacking the Handle contents also tends to fill memory with values that cause bus errors. This can flush out errors that otherwise might go undetected. Enormous amounts of time can be saved by such error detection. And if the bug gets into a shipping product, producing a revision to fix the problem can be very expensive.

Any system calls that allocate, dispose or otherwise manipulate memory are particularly appropriate for this kind of debugging code (e.g. `NewPtr`, `DisposePtr`, `NewHandle`, `DisposeHandle`, etc). Symantec's implementation of "debugging traps" now covers several hundred toolbox and operating system calls. While this may seem like a lot of work, the potential savings across multiple teams and multiple projects is huge. It is not uncommon for a programmer to spend a day or two tracking down an obscure crash that could easily be detected by a debugging trap. If you spend the time writing debugging traps just once, you'll get a steady dividend of increased productivity.

Note that use of debugging traps requires no source code changes; you continue to write:

```
DisposeHandle( theHandle );
```

You do not need to write:

```
DebugTraps_DisposeHandle( theHandle );
```

Instead, the C/C++ macro preprocessor takes care of substituting the call to `DebugTraps_DisposeHandle` instead of a direct call to `DisposeHandle`.

The elegance of this technique is that no changes are required to your source code, yet you have stringent debugging turned on for your code and the code that everyone else on your team writes! In fact, team members don't even have to know this technique is being used (except when it detects an error). The benefit of this, especially for legacy code, is enormous, because you get instant bug notification for any code which you compile. And when `DEBUG` is off, you pay no penalty whatsoever for this benefit.

Implementing a debugging traps scheme

To implement a debugging traps facility, you will need to do the following:

1. create source file(s) to contain the debugging traps, say "DebugTraps.c".

2. create a header file which contains macro definitions for the traps and function prototypes for the debugging calls ("DebugTraps.h"). When `DEBUG` is on, the macros redefine trap calls to vector to the debugging versions; when `DEBUG` is off, nothing is redefined and the code compiles normally. For example:

```
#if DEBUG
  #define BlockMove   DebugTraps_BlockMove

  pascal void       DebugTraps_BlockMove(const void *srcPtr,
                        void *destPtr, Size byteCount);
#endif
```

3. create an "off" file ("DebugTrapsOff.h"). This file #undefs all debugging traps. Such a file is used in a very few places (such as DebugTraps.c) so that the real trap may be called without recursion problems:

```
  #undef BlockMove
```

4. #include "DebugTraps.h" in any source files in which you want debugging traps. We include it in our project prefix so that all source files are subject to debugging traps. Obviously, the value is much greater to have it on for all files, rather than for just a few.

As you encounter new bugs having to do with making inappropriate system calls, take the time to add that knowledge to a debugging trap for that call. It will pay off handsomely, since you will never again make the same mistake—the debugging trap will notify you as soon as it happens! For example, a common mistake is to call DisposeHandle on a resource. There is no reason to ever make this mistake if you have a debugging version of `DisposeHandle` which flags an attempt to dispose a resource Handle.

When writing a debugging trap, it makes sense to place very stringent asserts on parameters of these debugging calls. Don't skimp; assert every single parameter as stringently as possible. Check for errors and flag any possible situation that could lead to a problem. Assume the client *will* pass bad parameters and will *not* check for errors, and flag those situations!

The most productive debugging traps are those that allocate, dispose, or otherwise manipulate memory or resources. The sample code included with this article contains debugging traps for memory. When you start to use it, consider implementing adding all resource manager calls right away.

Detecting memory leaks

We've now discussed how to write "debugging traps". This last section discusses how to detect memory leaks using the debugging traps facility. Detecting memory leaks is by far the most complicated technique to implement (but not to use). Only an overview can be given due to space considerations.

A common programming error is failure to release memory that is no longer used. Yet it is straightforward to eliminate this type of error forever, with almost no impact on your source code and no ongoing programming effort, even for new projects.

The leaks checking ("Leaks") module we use at Symantec requires the debugging traps facility discussed in this document. The approach is simple: all routines that allocate memory notify Leaks of the type, size and nature of the allocation. All routines that deallocate memory notify Leaks that the memory has been disposed. With debugging traps, this is trivial to implement: simply include calls to the Leaks code for memory that is allocated or deallocated. This includes routines such as `NewHandle`, `NewPtr`, and even routines such as `NewWindow`, `NewIconSuite`, `NewRgn`, etc. Corresponding disposal routines notify the Leaks manager when memory is disposed. Obviously, to track all possible types of leaks, you must write a debugging trap for any routine that allocates or disposes memory so that it can call the Leaks manager. Such routines include `NewWindow`, `NewIconSuite`, `NewRgn`, etc. C++ objects, though not handled in a debugging trap, are handled in a similar fashion: `operator new()` and `operator delete()` are replaced with versions that call the normal version, but also make calls to the Leaks manager.

Other types of memory allocation could be handled in a similar manner.

In addition to tracking whether memory is disposed, the Leaks manager verifies that it is being disposed of correctly. For example, you should not allocate a region with `NewRgn` and then dispose of it with `DisposeHandle`; instead you should call `DisposeRgn`.

Finally because Leaks code is debug code, it too disappears when `DEBUG` is off.

What the Leaks Manager does

The Leaks manager tracks memory allocation using a table of entries with one entry in the table for each allocated item. A table entry includes the following information for each allocated item:

- the size of the item;
- the kind (Handle, Ptr, object, etc);
- the call used to allocate the item (`NewHandle`, `NewPtr`, `operator new`, etc)
- a stack crawl;
- the name of the file containing the routine which allocated the memory

In addition, the Leaks manager maintains various other information, including statistics about how much memory of what kinds was allocated.

To use the leaks code, you first initialize it when you program starts up:

```
DebugLeaks_Init();
```

You then initiate a named "session"; all further memory allocations are remembered for this session:

```
DebugLeaks_StartSession( 1000, "\pmain");
```

To see if leaks have occurred, you stop the session. This reports all leaks that have occurred and disposes of the session:

```
DebugLeaks_StopSession( );
```

Here is how you would leak-proof your entire application:

```
void
main(void)
{
InitMac();
// initialize toolbox, etc

DebugLeaks_Init();
DebugLeaks_StartSession( 1000, "\pmain");

... initialize and run your program ...

DebugLeaks_StopSession( );
}
```

In Symantec's implementation, sessions may be nested; this feature is useful for localizing problems without having to modify code elsewhere. For nested sessions, the Leaks manager simply keeps a stack of sessions. Memory allocations are remembered in the current session; memory disposal checks each session in turn to find the item being disposed.

When you stop a session, the Leaks manager reports the total number of leaks, and reports information about each individual leak, including a stack crawl for the leak. This makes it straightforward in most cases to immediately determine where the offending item was allocated. (It may be harder to determine why it was not deallocated!).

Of course, there are always a few wrinkles. Sometimes you allocate items you know you will never dispose (say certain global data structures or objects). In this case, you don't want Leaks to report a leak. In such a case, you tell Leaks that the item is not a leak by calling `DebugLeaks_IgnoreItem` immediately after allocating the item.

A slightly more annoying problem is that programming frameworks permanently allocate items which they never dispose. To work around this, the routines `DebugLeaks_SuspendSession` and `DebugLeaks_ResumeSession` can be used to temporarily disable leaks tracking while the programming framework is initialized. In a few cases, you may have to take other steps so that the Leaks manager doesn't consider such allocations to be leaks. Although this requires a little work, it is straightforward. In some cases, Leaks can check the supplied source file name and ignore memory allocations from those files. In the future, I would like to see programming framework vendors provide proper disposal routines for their frameworks so that before program termination all memory can be disposed of properly.

Leaks API

The API to our leaks manager consists of a variety of routines, almost all of which are called exclusively from the debugging traps code to remember or forget a memory allocation. Just skim past these routines now, and refer to them later as needed. Also, refer to the source code templates for comments on what each routine should do.

```
void   DebugLeaks_Init( void );
void   DebugLeaks_Dispose( void );

void   DebugLeaks_StartSession( ulong maxItemsToTrack,
           ConstStringPtr sessionName);
void   DebugLeaks_StopSession( void );

void   DebugLeaks_SuspendSession( void );
void   DebugLeaks_ResumeSession( void );
void   DebugLeaks_AssertNotSuspended(void);

void   DebugLeaks_RememberHandle( Handle theHandle,
           DebugLeaksHowAllocated how,
           const char *srcFileName);
void   DebugLeaks_ForgetHandle( Handle theHandle );

void   DebugLeaks_RememberPtr( void *thePtr,
           DebugLeaksHowAllocated how,
           const char *srcFileName);
void   DebugLeaks_ForgetPtr( void *thePtr );

void   DebugLeaks_RememberAEDesc( AEDesc *theAEDesc,
           DebugLeaksHowAllocated how,
             const char *srcFileName);
void   DebugLeaks_ForgetAEDesc( AEDesc *theAEDesc );

void   DebugLeaks_RememberObject( void *object,
           ulong size);
void   DebugLeaks_ForgetObject( void *object );

Boolean  DebugLeaks_ItemIsRemembered( const void *item );

void   DebugLeaks_RefreshStackInfo( void *handleOrPtr );

void   DebugLeaks_DisposingHandle( Handle h );
void   DebugLeaks_DisposingPtr( Ptr p );

void   DebugLeaks_IgnoreItem( const void *item);
```

Suggestions For Implementing a Leaks Manager

The major housekeeping task of a leaks manager is maintaining a list of allocated items efficiently. In our implementation, when a session is started, you specify the maximum number of items that can be remembered in your call to `DebugLeaks_StartSession`. Memory is allocated at that time and released when the session is stopped. By allocating a fixed amount of memory per session, performance remains very high and the code stays simpler. If Leaks runs out of space to remember items, it complains. You then simply specify a higher limit, recompile, and try again (remember, this is debugging code). An improvement would be to have Leaks reallocate dynamically. In practice, this hasn't been necessary.

The other issue is that of tracking items in the list. Our approach has been to maintain a compact array of structs that contain information about allocated items. New items are added to the end of the list. When an item is removed, the last item in the list is put in its place. In this manner, the list stays compact and empty entries begin after the last in-use entry. For example, you could have a table of 1000 entries. If there are 232 items being tracked, then items [0,231] are in use and items [232, 999] are empty and available for use.

Searching the list is a simple array traversal and determining the number of items in it is trivial. Technically speaking, adding an item takes O(1), and removing an item takes O(n), where n is the total number of remembered items in all sessions. However, removed items typically are ones that were recently added, so they can usually be found right away, so finding items to remove is usually extremely fast. In any case such traversal is usually considerably faster than the original call to dispose of the item (such as `DisposeHandle`) and in our experience, has little effect on program speed. Speed of debugging code should not be an issue as long as it is reasonable. Sorting the list was considered, but is actually far less efficient than just using an unordered list.

Leaks checking is most useful when adequate information is provided to track down the source of the leak. Useful information includes a stack crawl, the type of item (object, Handle, Ptr), the size of the item, the routine which allocated the item (NewHandle, etc), the file containing the routine which allocated the item, etc. All of this information can be stored in a small amount of space and made available when a leak is detected.

The type, size and routine which allocated the item are all known at the time the item is allocated (see the routine `DebugTraps_NewHandle` in DebugTraps.c for an example). The file containing the allocating routine can be determined using the `__FILE__` preprocessor variable. `__FILE__` generates a null terminated C string which contains the name of the source file currently being compiled.

The stack crawl can be determined by traversing return addresses on 68K machines and link registers on PowerPC machines (on both machines, turning Macsbug symbols on allows you to display routine names, instead of raw addresses). To generate stack crawls you'll need to parse through the stack. On 68K machines, you'll want to make sure the generated code uses A6 stack frames with Macsbug symbols turned on. In PowerPC code, you'll need to turn on the "traceback" feature. Stack crawls are the most difficult piece of information to obtain. You may want to omit this information is you are not adept at assembly language or are unfamiliar with the details of how the stack works on 68K and PPC machines.

Finally, all of this information can be written to a log file, for later analysis. You can implement as little or as much of this as you find necessary; more is usually better because it makes it easier to track down the leak.

Leaks code can also be used to track the number, kind, and frequency of memory allocation. This information could be used to tune your program for better performance.

One note about storing file names: it is much more space-efficient to reference a string in an in-memory string list than to maintain a copy of the filename for each allocated item. For example, 10 Handles could be allocated in file "MyFile.c". To save space, store the string "MyFile.c" in a string list, and refer to it by index. In other words, don't store the entire string "MyFile.c" in every allocated item. Since you have only a limited number of source files, your list of file names will never get very large. This approach will drastically reduce space requirements.

To keep track of which routine allocated the item, you'll want to use an enum containing values for each routine that allocates items. In the debugging trap, you'll pass this enum value to the leaks manager. For example, in DebugTraps_NewHandle, you'd make the following call to the Leaks manager:

```
DebugLeaks_RememberHandle( h,
  kLeaks_NewHandle, srcFileName);
```

Your Leaks manager should also be robust enough to detect attempts to remember the same item more than once or to forget it more than once. Attempts to do so are probably bugs in your program and should be flagged.

Prevention

It is better to prevent bugs in the first place than to detect them later. Use modern programming techniques that reduce the chances of bugs. Discussion of such techniques is beyond the scope of this article. However, there are a number of good books that discuss these types of issues. Here are several recommended books by authors who have their heads on straight regarding good design and coding practices:

1. Code Complete by Steve McConnell, Microsoft Press 1993, ISBN 1-55615-484-4. *See http://www.microsoft.com/mspress/books/des/5-484-4a.htm (yes, the 'l' is missing).*

2. 50 Effective Techniques For C++ by Scott Meyers, Addison-Wesley Professional Computing Series ISBN 0-201-56364-9. *See http://www.aw.com/cp/meyers-effective.html.*

3. Design Patterns by by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Addison-Wesley Professional Computing Series 0-201-63361-2. *See http://www.aw.com/cp/Gamma.html.*

Conclusion

The techniques discussed in this article will detect many unnecessary and mundane bugs. As they say in Hollywood, "don't call us, we'll call you". In this case, that's a Good Thing! Let your bugs notify you when they occur and spend your energy elsewhere.

In our experience at Symantec, use of these techniques has contributed directly to faster and higher quality development. Time that formerly would have been wasted tracking down hard-to-reproduce problems can now be spent productively. Problems that went unnoticed in the past are now noticed—before shipping the product. All developers will benefit from the application of the techniques discussed in this article.

All of the techniques discussed in this article can be implemented by a good programmer in about a week. Make that effort just once and you'll benefit far into the future.

About the Sample Code

The sample code is provided with a Metrowerks CodeWarrior 9 project. The code has not been compiled with other environments but it should compile with little or no modification in other environments. The code does not require C++, but should compile fine under a C++ compiler.

Please note that when you run the sample project you *will* drop into the debugger. The intention is to demonstrate how certain asserts work. Make sure you have Macsbug installed!

Refer to the sample code *(available at http://www.mactech.com/-ed/wgi)* for more details on implementation of debugging traps and leaks checking. You should be able to use the debug traps facilility provided in the sample code immediately in all your projects with no modification. The Leaks manager portion is not implemented although the stub routines are called by the debug traps code. It should be fairly straightforward to implement the Leaks code to provide basic functionality. Data structures are suggested in DebugLeaks.c.

SYMANTEC TOP TEN

By Scott Morison, Symantec Technical Support

SYMANTEC.™

This monthly column, written by Symantec's Technical Support Engineers, is intended to give our readers technical information on using Symantec products.

Q: After upgrading my Mac from 64M to 136MB of RAM the Symantec Project Manager (v8.1) now barks at me with -37 error, saying, "The application Symantec Project Manager cannot start up because of an unknown error." What gives?

A: This error will occur while launching the Symantec Project Manager on any Macintosh possessing more than 110MB of RAM. There are two ways to fix this:

1) Call or send e-mail to Symantec Technical Support and request the SPM -37 patch.
   Phone: (541) 465-8470
   E-mail: support@devtools.symantec.com
2) You can make the necessary changes to the Symantec Project Manager yourself using ResEdit:
   • Make a copy of the Symantec Project Manager and open it in ResEdit.
   • Open the STR# ID 203 in the STR# resource.
   • Change the 2nd string of STR# 203 to read, <Options>
   • Change the 3rd string of STR# 203 to read, <Prefs>

Q: How do I send and receive data from the a serial port on my Mac?

A: This code sample walks through the basics of sending and receiving character data from a serial port:

[Thanks to Mark Y. Geschelin for the original code this is based on.]

```
#include <console.h>
#include <Serial.h>
#include <Devices.h>
#include <stdio.h>
#include <console.h>
#include <stdlib.h>
#include <Serial.h>
#include <Devices.h>

#define SERBUFSIZ 1024      // Define the Input buffer size to use
char *inbuf;                // pointer to input character buffer
short inRefNum, outRefNum; // Device driver Reference Number holders

////////////////////////////////////////
// Initialize the Serial Port  //
////////////////////////////////////////
OSErr InitializeSerialPort()
{
  OSErr   err;
  SerShk flags;
  Ptr buf;
  // Open Serial Drivers (note: Use ".BIn" and ".BOut" for Printer port)
  // assign Output and Input driver reference numbers
  if (err = OpenDriver("\p.AOut",&outRefNum)) return err;
  if (err = OpenDriver("\p.AIn", &inRefNum)) return err;

  // Initialize input and output drivers, and
  // assign basic communication protocols
  if (err = SerReset(outRefNum,baud57600 + data8 +
stop10+noParity))
    return err;
  if (err = SerReset(inRefNum,baud57600 + data8 +
stop10+noParity))
    return err;
  // Set up the serial input driver to use a buffer of size
SERBUFSIZ
  if(!(buf = NewPtr(SERBUFSIZ))) return MemError();
  if (err = SerSetBuf(inRefNum, buf, SERBUFSIZ)) return err;

  // Specify handshaking and cotrol info for the input driver
  flags.fXOn = false; // XOn/Xoff Output enabled?
  flags.fCTS = true;  // Using Clear To Send harware handshaking?
  flags.xOn = 0x11;   // Character for XOn
  flags.xOff = 0x13;  // Character for XOff
  flags.errs = false; // Abort Input requests if: Parity error
                      //     or: Hardware overrun
                      //     or: Framing error
  flags.evts = false; // Post event on CTS or Break status change
  flags.fInX = false; // XOn/Xoff Input enabled?
  flags.fDTR = false; // Using Data Terminal Ready flow control

  // Set driver to reflect settings
  if ( err = SerHShake(outRefNum,&flags)) return err;
  // Allocate input buffer; return reason on failure
  if (!( inbuf = (char *) NewPtr(SERBUFSIZ))) return
MemError();
  return noErr;   // noErr = 0
}

////////////////////////////////////////////////////
// Send a String to the Serial Port  //
////////////////////////////////////////////////////
void SendSerial(char *outString, long strLen)
{
   FSWrite(outRefNum,  &strLen, outString);
}
```

[Thanks to Andrew McFarland, Noah Lieberman and Levi Brown for their contributions.]

```
////////////////////////////
// Main //
////////////////////////////
int main(void)
{
  OSErr err;
  long count;
  char keyChar;

  csetmode(C_RAW,stdin);    // disable echo and line buffering for input

  if (err = InitializeSerialPort()) // Check for failure to initialize port
    printf("Serial initialization failed. Error = %d\n",err);
  else
  {
    SendSerial("ATX\r",5);        // Send ubiquitous Hayes reset
    keyChar = getchar();          // Get a character from stdin
    while (keyChar != 0x1B)       // Loop until escape key is pressed
    {
      if (keyChar > 0)            // Is there a character to send?
      SendSerial(&keyChar,1);     // Call SendSerial to send it.

      SerGetBuf(inRefNum, &count);
                                  // Is there anything in the Input buffer
      if (count)
      {
        FSRead(inRefNum,&count,inbuf);
                                  // Read all chars from Input driver
                                  // Send to console
        for (long i=0; i < count; putchar(inbuf[i++]));
      }
      keyChar = getchar();        // Get another character from stdin
    }
  }

  // Clean up: Reset Ports, return pointer
  if(outRefNum) CloseDriver(outRefNum);
  if(inRefNum) CloseDriver(inRefNum);
  if (inbuf)  DisposPtr(inbuf);

  return EXIT_SUCCESS;
}
```

Q: How do I load and play a sound from a 'snd ' resource in my Symantec C/C++ or Pascal application?

A: Here's an example of how to do just that, in both C and Pascal.

```
#include <Sound.h>

void CallSndPlay(void);         // Function Prototype
void CallSndPlay()
{
  Handle mySndHandle;           // handle to an 'snd ' resource
  SndChannelPtr mySndChan;      // pointer to a sound channel
  OSErr myErr;

  void DoError(OSErr);          // prototype for your DoError function

  mySndChan = nil;              // Initialize channel ptr for error checking
  mySndHandle = GetResource('snd ', mySndID);
                                // Read in 'snd' resource from resource

  if ( mySndHandle != NULL ) // Check for NULL handle
  {
    myErr = SndPlay (mySndChan, mySndHandle, TRUE);
    if ( myErr )
      DoError(myErr);           // You define the function, "DoError."
  }
}

int main()
{
  InitToolbox();                // Function you get to define.
  CallSndPlay(9000);            // Play 'snd' resource ID 9000
}
```

And the same snippet in Pascal would look like this:

```
program mySound;
uses Sound;

procedure CallSndPlay (mySndID: integer);

  { Be sure to add the file sound.p to your project }

var
  mySndHandle: Handle;     { handle to an 'snd ' resource }

  mySndChan: SndChannelPtr;     { pointer to a sound channel }
  myErr: OSErr;

begin  { CallSndPlay }

  mySndChan   := nil;      { Initialize channel ptr for error
checking }
  mySndHandle := GetResource('snd ', mySndID);
                 { Read in 'snd' resource from resource }

  if (mySndHandle <> nil) then   { check for a nil handle }
  begin
    myErr := SndPlay(mySndChan, mySndHandle, true);
    if (myErr <> noErr) then
      DoError(myErr);      { You define the procedure,
"DoError." }
  end;

end; { CallSndPlay }


begin  { Main }
  InitToolbox;         { You need to add this procedure
yourself. }
  CallSndPlay(9000);     { Play 'snd' resource ID 9000 }
end. { Main }
```

Q: How do you view the contents of an array in the Symantec v8.1 Debugger?

A: Highlight the array in the debugger source window and hit Command-D, or just type the name of the array into the data view window.

- Select Address rather than Pointer from the Data menu.
- Turn down the hierarchical arrow to display array contents.

Q: How do I disable the debugging call outs that are embedded in the native Exception handling routines in Symantec C++?

A: Comment out the debugging **#defines** in, **TCL #includes.cpp**. Then re-precompile your headers.

Find the lines:

```
#define TCL_DEBUG           // include debugging code, TCL_ASSERT, etc.
#define BR_DEBUG            // if debugging BEL
#define TCL_BREAK_CATCH     // enter debugger on catch_all_()
#define TCL_BREAK_FAILURE   // enter debugger on Failure()
#define TCL_BREAK_ASSERT
// enter debugger on TCL_ASSERT fail (2.0.5) and comment them out:
#define TCL_DEBUG           // include debugging code, TCL_ASSERT, etc.
#define BR_DEBUG            // if debugging BEL
#define TCL_BREAK_CATCH     // enter debugger on catch_all_()
#define TCL_BREAK_FAILURE   // enter debugger on Failure()
#define TCL_BREAK_ASSERT    // enter debugger on TCL_ASSERT fail (2.0.5)
```

Q: How do I convert projects that use MetroWerks proprietary .lib format libraries such as **AEGizmos.lib**, with Symantec C++.

A: We have recently built a MW .lib library format translator that allows you to simply drop a CodeWarrior v8 .lib format library into your Symantec C/C++ project allowing you to call any routines defined therein. This new translator will be available on our 8.0r6 CD coming this Fall.
If you would like to obtain this translator prior to the release of the CD, feel free to contact Symantec Technical Support:
Phone: 541/465-8470
E-mail: support@devtools.symantec.com

Q: Using Cafe, I have derived class B from class A. Since class B does not have a constructor, how do I pass the parameters to class A?

A: Whenever a class is instantiated, a default constructor (no parameters) is implicitly called if you do not create one explicitly. You need give class B a constructor which will receive the parameters and then pass them on to A via the **super** keyword.

```
class B extends A
{
  public B(double  aParameter)
  {
    super(aParameter);
  }
}
```

Q: Using Cafe how can I make a **Component** that is drawn to the screen observable since I cannot derive from both **Component** and **Observable**?

A: Say you want to make the cells in a Java spreadsheet both **Observable** and **Observers** so that you can perform distributed calculations based only on the items that changed, however, the items in your "table" need to be text fields so that the user can input the data. You need to do the following: 1) Create a Cell class that derives from **Observable** and implements **Observer**. 2) Create a **SmartTextField** class that derives from **TextField**, and make a data member that is a **Cell**. Now when data is entered in your **SmartTextField** you can process events in the **SmartTextField** and set variables in the **Cell** data member, call its **notifyObservers** method, etc.

Note: you can also have a reference in the **Cell** to the **SmartTextField** that contains it so that you can set the text:

```
class Cell extends Observable implements Observer
{
  double  curValue;
  double  oldValue;
  double  delta;

  SmartTextField  theText;

  public Cell(SmartTextField  aText)
  {
    theText = aText;
  }
  public void Update(Observable o, Object arg)
  {
    oldValue = curValue;// We are using a model where the cell is both an
                        // observing and observed so if this method is being
                        // called then a cell that this object
                        // observes has changed.
    curValue += ((Cell)o).delta; // calculate delta
    theText.setText(""+curValue);
                        // set the SmartTextField to the new Value
    super.setChanged();
    notifyObservers();
    super.clearChanged();
  }
}

class SmartTextField extends TextField
{
  Cell    theCell;

  public SmartTextField(String someText, int width)
  {
    super(someText, width);
    theCell = new Cell(this);
  }

  ////——— handleEvent ———/////
  public boolean handleEvent(Event evt)
  {
    // when the cell gets the focus, save its value
    // so we can  check if it has changed when it loses the focus
    if(evt.id == evt.GOT_FOCUS)
    {
      ((SmartTextField)evt.target).selectAll();
      ((SmartTextField)evt.target).theCell.oldVal =
        Double.valueOf(((SmartTextField)evt.target)
        .getText()).doubleValue();
      return false;
    }

      //lost the focus check to see if the value changed and deal with it
    if(evt.id == evt.LOST_FOCUS)
    {
      //get the value of the current cell
      ((SmartTextField)evt.target).theCell.curVal =
        Double.valueOf(((SmartTextField)evt.target)
        .getText()).doubleValue();

      //calculate the delta
      ((SmartTextField)evt.target).theCell.delta =
        ((SmartTextField)evt.target).theCell.curVal -
        ((SmartTextField)evt.target).theCell.oldVal;

      //if the delta is non zero, i.e. the value was changed
      if ( ((SmartTextField)evt.target).theCell.delta != 0 )
      {
        ((SmartTextField)evt.target).theCell.setChanged();
((SmartTextField)evt.target).theCell.notifyObservers();
        return true;
      }
      else
        return false;
    }
    else
      return false;
  }
}
```

By Christopher Haupt

Implementing SMTP with PowerPlant

Create a simple Internet mail sender using PowerPlant's network classes

Introduction

Since the Internet's explosive growth in the early 1990's, a variety of new and interesting tools have been developed to explore its resources. Without doubt, the greatest attention has been placed on the World Wide Web; browser battles are a constant focus of the media. Many new users want to be able to cruise about, exploring the vast resources open to them. This attention momentarily diverts people from the fact that the most frequently used tools on the Internet are file transfer (FTP) and electronic mail (often SMTP/POP) programs.

If you consider the experiences you have today on the Web, you will note the solitary nature of that activity. Only now are sites becoming aware of the community building abilities of this new media. By and large, people want to communicate with each other, and not just enter a ghost town of information.

Over the past few years, my company has focused on this community building aspect of net use. In particular, we are interested in building tools that allow kids to meet one another and form relationships. In the beginning, we hope to encourage these relationships through forming "pen-pals", or email friends. To this end, we are exploring the implementation of simple Internet mail enabled tools. From this research I present this introduction to the primary mail sending protocol, the Simple Mail Transport Protocol (RFC821), and explore a simple implementation of this protocol using Metrowerks' PowerPlant network class library.

This article does not provide an introduction to TCP/IP development, but rather the higher level mail protocol. Several good references exist on TCP/IP, of which I recommend (Stevens 94), (Comer 91), and (Tannenbaum 81) as useful additions to your library.

The code presented in this article meets several simple requirements. It implements SMTP using the current PowerPlant network classes. It provides the ability to send "one-shot" email messages–it doesn't store messages in a mailbox. Additionally, it will not receive messages, although implementing a mail receiver using the Post Office Protocol (RFC1725) can be done using this code as a guide. By using the latest PowerPlant, we also get the ability to dynamically switch between MacTCP and OpenTransport without additional code.

The Simple Mail Transport Protocol (SMTP)

The Simple Mail Transport Protocol was designed to be an easily implemented, reliable mechanism for moving messages from one trusted host to another. This article includes an overview of the protocol, but the definitive specification is (RFC821). (Stevens 94) is an excellent treatment of this material.

SMTP is specified independent of a transport service (it can run over any type of network with a reliable transport layer). This paper describes an SMTP implementation using TCP, which is the most common transport medium in use today for SMTP on

Christopher Haupt is the chief technical officer of CyberPuppy Software, Inc. He is the architect for the first children's Internet collaborative environment, PigMail™. In his spare time, he writes papers for MacHack, creates multiplayer net games, and flies kites. He can be reached at cfh@cyberpuppy.com.

microcomputers. SMTP is assigned to the permanent TCP port 25.

The SMTP specification describes a lock-step protocol in which the sender and the receiver transmit specifically formatted ASCII messages to one another, awaiting a response before continuing. At a high level, the SMTP architecture can be described by a simple finite state machine which contains three main states (see Figure 1). The machinery is either checking reply codes, sending new commands, or sending the message contents. In smart implementations, an error code does not necessarily force a disconnect, although it is illustrated and implemented this way in this article.

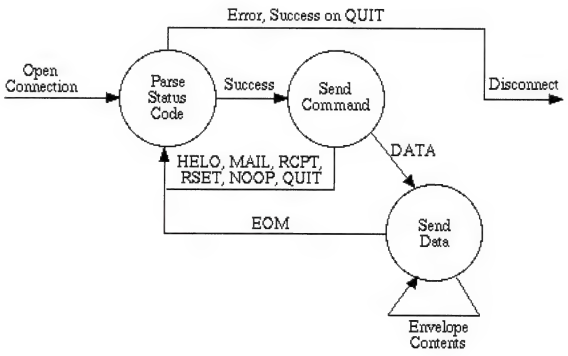


Figure 1. SMTP Finite State Machine

SMTP defines a small required command set, with several optional commands included for convenience purposes. Table 1 shows the minimal set required for an SMTP sending client.

HELO - Initial State Identification
MAIL - Mail Sender Reverse Path
RCPT - One Recipient's Forward Path
DATA - Mail Message Text State
RSET - Abort Transaction and Reset all buffers
NOOP - No Operation
QUIT - Commit Message and Close Channel

Table 1. Minimum SMTP Command Set

Commands may have zero or more parameters. Commands and their parameters are issued as ASCII plain text strings. A command is terminated with a carriage-return, line-feed (<CRLF>) pair. Commands do not span lines. The termination pair completes the command line when it is encountered. For instance, the command to identify the sender of a mail message would be sent as MAIL FROM:<cfh@cyberpuppy.com><CRLF>.

Acknowledgment messages are formed by a three digit return code, followed by optional text. The three digits represent error and success codes. A typical success message would appear as 250 Requested mail action okay<CRLF>. Note that, within an acknowledgment message, only the first three digits are significant. The textual portion of the reply messages is for human understanding and can contain any text. Messages are grouped by meaning by using the first digit as a key.

Messages beginning with a "2" are success messages, "3"'s are error codes, etc.

Normally, the acknowledging process will send one reply message per command. Each reply is terminated with the standard <CRLF> token. It is possible that more than one acknowledgment message may be sent, and this is not prohibited by the protocol specification. You should consider that some servers may generate more than one line of response and handle that case accordingly–this occurs most frequently with message 220, the service ready message transmitted on startup from the receiver when the sender initiates a connection. If you aren't careful, this can throw your state machine off. The SMTP specification states that multiline responses should include a hyphen ("-") immediately following the result code of each intermediate status code. The final result line is formatted normally, without the hyphen.

A typical SMTP session can be characterized as shown in Figure 2 and described here. The sender ([S]) opens a two-way channel to the receiver ([R]). The receiver can be the final destination or an intermediate node, described in the message's path explicitly, or implicitly by network routing tables. At connect time, both hosts are in the Initial state. [R] sends an acknowledgment that the channel is open. [S] sends a HELO message, identifying itself to the receiver. Note that authentication is not required, so it is very easy to spoof sender

IP addresses; SMTP is not a secure messaging protocol. [R] sends back a success or error message, possibly denying access to the sender. If the HELO was successful, both sides are now in the Send Command/Response state. [S] sends a MAIL command describing the sending party's fully qualified reverse path. [R] acknowledges the successful receipt of the path and clears all of its transaction buffers. [S] sends one or more RCPT commands describing the forward path of recipients of the mail message, one recipient per line. [R] accepts or rejects each address.

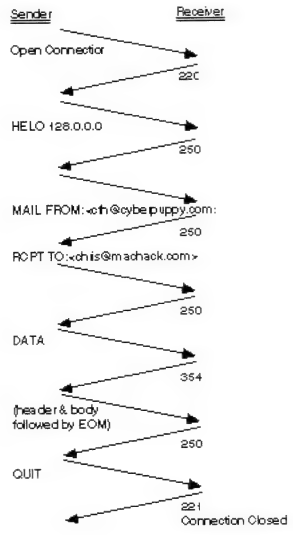


Figure 2. Typical SMTP Transaction Flow

[S] now sends a DATA command, instructing the receiver that all following data is the actual mail message, thereby putting the transaction in the Send Data state. Transmission of the message text completes when the end-of-message (EOM) sequence is sent (a <CRLF>.<CRLF> triplet, which looks like a period alone on a line). This raises the question, "what if the message contains the EOM sequence?" Data transparency is achieved by stuffing any instance of the EOM sequence occurring within the body of a message with a period "." character prefix. The receiver checks each line for a leading period and removes it before buffering the data. Only when [R] detects the "real", tailing EOM, does it send an acknowledgment.

[S] sends a QUIT command to place the transaction in the Commit state. [R] acknowledges the command and closes the channel. It then delivers the message to the recipients' mailboxes or forwards the message on to the next server in the recipients' forward paths.

You will note that the SMTP protocol does not handle any of the fields you would associate with a standard mail message (fields such as Subject:, Reply-To:, etc.). These fields, which make up a message that conforms to (RFC822), are built and parsed by the mail handling agent on either end of the SMTP transaction. SMTP treats the mail message in an opaque manner, sending the headers and message body all at once during the Send Data state. The SMTP code only peeks at the message to ascertain EOM transparency conditions. SMTP places the path information generated in a transaction at the front of the message contents. This area is often called the envelope.

Implementing SMTP in PowerPlant

To implement a simple SMTP client for the PigMail project, I chose to create a rudimentary mail editor and tie it to the SMTP code by using a LSingleDoc derived class and its associated window member. The implementation started out using a threaded approach. Before long, debugging of the PowerPlant network classes in the older versions of CodeWarrior bogged the project down. I muttered "Keep It Simple Stupid" to myself a couple of times and created the very simple, event-loop based asynchronous version which is presented here.

A threaded implementation is actually not much more difficult to construct, but it does add some complexity to the discussion at this introductory level. Because SMTP is a simple problem domain, it is a great opportunity to experiment with threading. You could implement the entire SMTP state machine as one thread, to which you hand off all data and let it rip. Or, you could be creative and implement a two thread approach and play with the Producer/Consumer model of cooperative processes (Silberschatz 92). I tried both, and while they work fine, they violated my KISS requirement. The most important thing I learned with these experiments was the danger of mixing threads which operate with different PowerPlant drawing contexts; talk about major view foci problems!

The simple mail sender class displays a window which contains a number of text edit fields: SMTP host, sender address, destination address, subject, and message body. It also contains a button to send the message when done and a status field. Figure 3 shows a picture of the simple interface.

Figure 3. Simple Mail Editor

In the event-loop/asynchronous handling implementation below, I began by creating a LSingleDoc, LAsyncClient class similar to the one shown in Listing 1. To increase your understanding in the following walk-through, you may want to

refer to the sample source code that is supplied with this article's distribution.

Listing 1: SMTPSenderDoc.h

SMTPSenderDoc

This class implements all of the machinery necessary to experiment with the SMTP protocol using the PowerPlant network classes (in async/eventloop mode)

```
class SMTPSenderDoc : public LSingleDoc,
                      public LListener,
                      public LAsyncClient
{
public:
                    SMTPSenderDoc(LCommander *inSuper,
                          const LStr255 &inTo = "",
                          const LStr255 &inSubj = "");
  virtual           ~SMTPSenderDoc();

  virtual  void   ListenToMessage(MessageT inMessage,
                                  void*ioParam);
  virtual  void   Connect();
  virtual  void   Disconnect();
  virtual  Boolean  IsIdle();
  virtual  Boolean  AllowSubRemoval(LCommander* inSub);

protected:
  virtual  void   HandleAsyncMessage(const LAsyncMessage&
                                          inMessage);
      void   BuildSessionWindow(void);
      void   SendMailMessage(void);
      void   RunMachine(char *inDataBuffer,
                        Uint32 inDataSize);
      void   SendHELO(Boolean inUseShort = false);
      void   SendQUIT(void);
      void   SendRSET(void);
      void   SendNOOP(void);
      void     SendMAIL(void);
      void     SendRCPT(LStr255 &inRecipient);
      Boolean  SendNext(void);
      void     SendDATA(void);
      void     SendBody(void);
      void     SendHeader(void);
      Boolean  ParseReply(char *inBuffer, Uint32 inBufferLen,
               Uint32& inPos);

  MailPreferenceTypeH mMailPrefs;
  LStr255       mTo;
  LStr255       mSubject;
  Handle     mBody;
  LEndpoint*    mEndpoint;
  LCaption*     mStatusPane;
  Int32         mMachineState;
  Int32         mLastCode;
  Int32         mCurPos;
  Int32         mMachineReplyState;
  char       statusBuffer[8];
};
```

The constructor initializes all member data, and calls the ::BuildSessionWindow() member function to create the interface. The To: and Subject: fields are filled with optional data supplied by the caller of the constructor.

At this point, control rests within the standard PowerPlant event mechanism, and the user can interact with the editor. When her message is done, she presses the Send button, and away we go.

The SMTPSenderDoc class receives the button message via its ::ListenToMessage() method. Here we call the ::SendMailMessage() method. ::SendMailMessage() extracts the data from the UI and initiates a connection.

The SMTPSenderDoc::Connect() method makes use of a wonderful PowerPlant object called the UNetworkFactory. This object allows you to use the best transport mechanism installed at run time. It will automatically switch between OpenTransport and "Classic Networking" (a.k.a. MacTCP) depending upon which is active at the time the UNetworkFactory is called. We create an asynchronous endpoint object that uses the event-loop to receive incoming asynchronous messages. An endpoint is simply an object that represents one-half of the communication link.

After creating the endpoint, we bind it to a network address. We specify the address information for the originating host. When the ::Bind() operation completes, we connect to the SMTP server host. Listing 2 shows the connection sequence.

Listing 2: Connect Method

SMTPSenderDoc::Connect()

The connect method requests an asynchronous endpoint from the network factory which automatically will select the correct networking mechanism (OpenTransport or MacTCP). We then try to Bind to a local address.

```
void SMTPSenderDoc::Connect()
{
  mEndpoint =
    UNetworkFactory::CreateTCPEndpoint(
              UNetworkFactory::Asynchronous(this));
  ThrowIfNil_(mEndpoint);
  // Initialization: Bind to any local port
  LInternetIPAddress address(0, 0);
  mEndpoint->Bind(address); // when this completes,
                            // finish making connect
                            // in HandleAsyncMessage
}
```

The asynchronous networking mechanism in PowerPlant is very easy to use. When network commands complete, or incoming messages are received, the networking classes call your LAsyncClient object's ::HandleAsyncMessage() method. Here you can crack the incoming message and dispatch to your various handlers. Listing 3 shows how simple the ::HandleAsyncMessage() dispatch mechanism can be.

When we are establishing the initial connection, as soon as we are notified that the connection is created, we set our endpoint to be in auto-receive mode. This endpoint mode automatically issues a receive command on your connection, thereby catching all data that is sent to your client without needing to explicitly issue receive commands.

In the SMTPSenderDoc code, whenever we get something from the SMTP server, we send that in to our SMTP state machine (the ::RunMachine() method). ::RunMachine() alternates between parsing incoming messages for their response codes and sending the next appropriate SMTP command.

Listing 3: HandleAsyncMessage Method

SMTPSenderDoc::HandleAsyncMessage()

This method is called by the notifier mechanism of PowerPlant whenever we receive an asynchronous response to one of our earlier requests. We will dispatch according to the message type. This is the meat of async messaging in PP and is really simple! Note that there isn't a lot of heavy error checking in this sample.

```
void SMTPSenderDoc::HandleAsyncMessage(const LAsyncMessage&
                                          inMessage)
{
  switch (inMessage.GetMessageType()) {
    case T_DISCONNECT:
    case T_ORDREL:
```

```
        mEndpoint->AcceptDisconnect();
        // fall through as the connection is closed
        // at the other end and we won't necessarily
        // get the DISCONNECTCOMPLETE when we issue an
        // AcceptDisconnect instead of a Disconnect
      case T_DISCONNECTCOMPLETE:
        delete this;
        break;
      case T_BINDCOMPLETE:
        if (noErr == inMessage.GetResultCode())
        {
          LInternetDNSAddress remoteAddress(mSMTPHost,
                                            kSMTPPort);
          mEndpoint->Connect(remoteAddress);
        }
        break;
      case T_CONNECT:
        if (inMessage.GetResultCode() == noErr)
          mEndpoint->AutoReceive();
        break;
      case T_DATA:
      case T_EXDATA:
        LDataArrived* data = (LDataArrived*) &inMessage;
        if (data->GetDataSize())
          RunMachine((char *) data->GetDataBuffer(),
                     data->GetDataSize());
        break;
    }
}
```

Listing 4 shows part of the ::RunMachine() method, which is an example implementation of the SMTP state machine described above. This implementation is a little unusual, and probably a little less clear, because its external switch statement jumps between result codes, while the inner conditionals branch on the actual state of the system. This code folds the alternating Reply/Send states together. Most of the time, the machine will be receiving state code 250 (success) and staying within the first case. Other status cases cover initialization, rundown, and error conditions.

Listing 4: RunMachine segment

SMTPSenderDoc::RunMachine()

RunMachine is the state machine for the SMTP protocol. In this implementation, our outer switch detects the last response of the SMTP transaction while the inner switches select the proper next state sequence. This Listing is a subsection of the entire method.

```
void SMTPSenderDoc::RunMachine(char *inDataBuffer,
                               Uint32 inDataSize)
{
  Uint32 thePosition = 0;
  Boolean done = false;

  // this mechanism provides an intermediate buffering in
  // case all of the reply hasn't arrived yet, we wait until
  // the line terminator is received before parsing
  while (!done && thePosition < inDataSize)
    if (ParseReply(inDataBuffer, inDataSize, thePosition))
    {
      switch (mLastCode) {
        case 251:   // ok, but non-local user
        case 250:   // success
          switch (mMachineState) {
            case eGreetingLong: // HELO was successful
            case eGreetingShort:
              SendMAIL();
              mMachineState = eMailSender;
              break;
            case eMailSender:    // MAIL was successful
              if (SendNext())
                mMachineState = eMailDestination;
              else
                mMachineState = eMailInitiateData;
              break;
            case eMailDestination: // RCPT worked, more?
              if (!SendNext())
                mMachineState = eMailInitiateData;
              break;
            case eMailInitiateData:  //recipients accepted
              SendDATA();
              mMachineState = eMailBody;
              break;
            case eQuitting: // data accepted
              SendQUIT();
              mMachineState = eDisconnecting;
              break;
            case eDisconnecting: // n/a
              break;
            default:
              Assert_(false); // this is for debugging
              break;
          }
          break;
/*
...error cases and intermediate data case removed...see sample source
*/
        }
    }
}
```

::ParseReply() collects the return information from the server and breaks out the result code. It discards the extra textual information. The **Send** methods simply format the corresponding commands with any parameters and push them out the endpoint. Note that the ::SendNext() method actually parses the To: field's data to allow for more than one destination address. In this way, the user can specify a comma delimited list of mail addresses. SMTP allows one forward path per **RCPT** command, so we have to cycle through the *n* addresses sequentially and send multiple **RCPT** commands if more than one recipient is specified.

When we get to the Send Data state, we begin by formatting and sending a (RFC822) header. The header minimally includes return path information, subject, recipient address information, and a properly formatted date field. Other fields are optionally appended to the header. The client then sends each line of the message body, testing each line for instances of the **EOM** sequence and properly byte-stuffs those lines.

At the end of the message body, an **EOM** is sent. Assuming that the server has accepted the transaction up to this point, we send a **QUIT** command, which commits the transaction.

The **QUIT** causes the SMTP server to send an acknowledgment and close the TCP channel from that end. The **LAsyncClient** object receives a **T_ORDREL** message requesting an orderly shutdown of the endpoint. The endpoint accepts the disconnect request and then deletes itself.

Assuming that no error messages were encountered, we just sent an Internet mail message via SMTP!

Implementation Problems

During this exercise, I encountered several gotchas with PowerPlant. Here I will try to explain them. Note that some of these issues are expected to be fixed by the PowerPlant release for CodeWarrior 10. The bugs and problems have been reported to Metrowerks and are described here in case you are using an older version of PowerPlant.

Endpoints in their current implementation are tricky beasts.

Developer Central™

Sponsored by

Apple Computer, Inc.

and

For Macintosh Programmers & Developers

MacTech™ MAGAZINE

Macworld Expo/San Francisco • January 7 – 10, 1997
Moscone Center – North

If you are developing Internet, multimedia, custom applications, or engineering solutions for the Mac® OS and other Apple® platforms, then you need to visit Developer Central™ — your all-in-one source for the latest tools for Mac OS development.

Whether you want to jump-start your knowledge of Mac OS development, or come up-to-speed on new tools or technologies, Developer Central will give you the inside scoop on Mac OS development by putting you in front of representatives from Apple Developer Relations and third-party tools developers. You'll even access the latest training and developer support programs offered by Apple.

At Developer Central, you can…

- Talk directly to representatives from Apple and third-party vendors
- Demo developer products and tools
- Attend sessions in the Developer Central theaters
- Get the best deals on tools, training, and developer resource materials

All Great Software Wears This Face

Mac OS

So, no matter if you're developing a departmental system, using the Mac OS in science, developing plug-ins for the Internet, or planning the next great Mac OS application or multimedia title,

Developer Central
The Source for Macintosh Development

Sponsored by Apple Computer, Inc. and MacTech Magazine

One problem with the asynchronous model is that you can get unusual dependencies that are not normally expected. One of the great current mysteries of the PowerPlant networking classes is when to properly destroy an endpoint. The asynchronous messages which tell an LAsyncClient what is happening are allocated out of a pool of memory created by the LEndpoint's Notifier object (most of the time. Actually there was a "bug" in that some LNetMessage objects were created from the endpoint's pool in CW8.) The notifier is destroyed by the endpoint when the endpoint is destroyed. Unfortunately, if you delete your endpoint from within ::HandleAsyncMessage() when you receive a message–such as T-DISCONNECTCOMPLETE–you will kill the memory pool from which the message is currently allocated. This causes problems when the message call stack pops back to the messages originating method, and then tries to delete itself again Boom!

The destructor in the sample code defers the deletion of the LEndpoint object. In the example, a thread is spawned to handle the deletion. This is clearly a work around, and an official solution may exist in a future version of PowerPlant.

A second problem can occur due to overflow problems on sending data. The current endpoint implementations of the Send method do not notify you if the outgoing data has caused an overflow condition. This can happen if you are relying on the auto-send mechanism which copies your data to an intermediate buffer. If you are generating data to go out more quickly that it can be sent, or if you try to send chunks larger than the pool can accommodate, the send will fail silently. There is the beginning of a mechanism to implement a notification of this error, but it is not complete in the CodeWarrior 9 release. You will need to apply a work around in your own code and/or modify the class to patch this up. In my sample code, I simply return the size of the buffer sent or the result code by changing the Send methods to make the data size parameter a reference copy (e.g. void LMacTCPEndpoint::Send(void* inData, Uint32& ioDataSize, LNotifier* inNotifier)).

Another significant gotcha is encountered when you implement a scheme in which you take very small pieces of data out of the incoming data stream. By making repeated calls to the endpoint's Receive method with a buffer size of one–or other small sizes–you will quickly run out of pool space. Or so it will seem. On closer examination, you will note that there is sufficient space within the pool, but it is impossible to allocate space for your receive request. This fragmentation of the endpoint pools can be avoided by using all data immediately when you get a T-DATA or T-EXDATA message.

If you must receive data in extremely small pieces, you can implement a more sophisticated free block coalescing algorithm, or some kind of intermediate buffering scheme. Using all of the data immediately appears to be the most efficient mechanism at this time. This problem is expected to be fixed in CodeWarrior 10.

A final problem with the networking classes is a collection of memory leaks. There are some instances of exceptions being raised before deleting memory allocated from a pool. See LMacTCPEndpoint::Receive() for an example. These leaks are being cleaned up in the CodeWarrior 9 and 10 releases.

Conclusion

The goal of this article has been to get you started on your own experiments with the networking classes of PowerPlant in general, and SMTP in particular. With the provided code, you should be able to add mail sending via SMTP to your project in short order. Internet programming is not as mysterious as you may think, and armed with this simple example, you can tackle other interesting Internet protocols.

As the Internet's popularity grows, your task of writing classes like this one will become much easier. You can be sure that class libraries like PowerPlant will have the functionality described here as standard features in the not too distant future.

Bibliography and References

Comer, Douglas E.. *Internetworking with TCP/IP Volume I: Principles, Protocols, and Architecture.* Prentice Hall, Englewood Cliffs, New Jersey. 1991.

Crocker, D.H. "Standard For The Format of ARPA Internet Text Messages," ARPANET Request for Comments, No. 821. SRI International: Menlo Park. August 1982.

Postel, J.B. "Simple Mail Transfer Protocol," ARPANET Request for Comments, No. 822. SRI International: Menlo Park. August 1982.

Myers, J.. "Post Office Protocol - Version 3," ARPANET Request for Comments, No. 1725. SRI International: Menlo Park. November 1994.

Silberschatz, Abraham, Peterson, James C., et. al.. *Operating System Concepts, 3rd Edition.* Addison-Wesley Publishing Company, Reading, Mass.. 1991.

Stevens, W. Richard. *TCP/IP Illustrated, Volume 1: The Protocols.* Addison-Wesley Publishing Company, Reading, Mass.. 1994.

Tannenbaum, Andrew. *Computer Networks.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey. 1981.

NEWSBITS

By Jessica Courtney

Sun And Apple Announce Alliance

Apple and Sun are joining together in a far-reaching strategic alliance. It is their common goal to provide powerful, flexible and media-rich Internet solutions that combine Sun's and Apple's expertise.

They will work together with Sun to leverage the traditional strengths of their respective companies — Sun's leadership position in the enterprise Internet/intranet server area and their industry-leading Java technologies, and Apple's legacy in ease of use and multimedia technologies, with QuickTime, the OpenDoc developer framework and Open Transport networking technology.

Sun and Apple will enhance their operating environments to provide a seamless bridge between Sun Solaris servers and Mac OS clients. This means customers will be able to tap into powerful Solaris-based environments from Macintosh clients, realizing the benefits of robust, high-performance networking with full connectivity, advanced network management and complete network security.

Apple will work together with Sun to develop cost-effective, cross-platform and customizable application solutions by enhancing the integration of OpenDoc and Java Beans, making compound document development easier. Sun will enhance JavaMedia to fully support QuickTime

Sun And Apple Announce Alliance Press Release
<http://product.info.apple.com/pr/press.releases/1996/q4/960918.pr.rel.sun.html>
Sun And Apple Announce Alliance Q & A
<http://web.apple.com/Areas/NewsArchive/199609/applesunqa>

Metrowerks Launches New CodeWarrior 10

New CodeWarrior IDE improves overall ease-of-use, offers Direct-to-SOM support and new visual support for Java. CodeWarrior Gold provides support for software development in four different programming languages (C/C++, Object Pascal and Java™), for five different operating systems (Mac™ OS, Windows® 95/NT, Be OS™, Magic Cap™ and PowerTV™ OS), and four different microprocessors (68K, PowerPC™, X86, MIPS®), all from one intuitive development environment.

Two new features have been added in Version 10 that make developing with Java considerably easier. A new release of Metrowerks Constructor now allows for visual development of graphical user interfaces, or GUIs, with Java. The applet viewer is now conveniently supported from within the IDE so users can run HTML files without opening a separate application.

Other new features in the Java tools include version 1.0.2 of the JDK from Sun Microsystems®, a new optimize option in the Java compiler, a new Java utility called CodeWrangler, which allows developers to view and alter the content of uncompressed Java zip files, disassembly support in the IDE, plus "thrill-seeker" (or alpha) versions of Metrowerks' Just-In-Time (JIT) compiler and a new native Java compiler.

C/C+ compilers now offer support for Direct-To-SOM language extensions for OpenDoc development, and a stable debugger for Copland, Apple's version 8 of the Mac OS, is also included. CodeWarrior Gold 10 is priced at $399 and is available through Developer Depot (800) MACDEV-1 or from Metrowerks at (800) 377-5416. <http://www.metrowerks.com>

Apprentice 5 is shipping!

Apprentice 5 contains dozens of libraries and classes (including a complete suite of PowerPlant and Think Class Libraries), from graphics and sounds to menu management and TCP/IP communications. Many of the libraries include complete source code.

Apprentice 5 retails for $35, but as a special offer to MacDev-1 readers, you can buy a copy of Apprentice for only $25 including shipping (800) 835-5514
<http://www.celestin.com/macdev1.html>

Capite Libro 1.0

Capite Libro is very well suited for ReadMe-documents, since the compiled documents (i.e., applications) can run on anything from a Mac Plus with System 6.0.5; and the program files are small (and fast). Furthermore, it's very easy to update your ReadMe-documents with new compilations.

Capite Libro 1.0 requires a Macintosh Plus or better MacOS compatible computer with System 6.0.5 or later and 600 K of free memory. Price: $90. For questions, please contact Torbjoern Larsson Fax: +46-18 212 673-42 Vox: +46-18 212 673-41

Fantasm V4.10 and PowerFantasm V4.10

Lightsoft announces the release of both Fantasm and PowerFantasm versions 4.10. PowerFantasm and Fantasm are complete assembly language development systems for Macintosh™ computers.

Two versions are available. Fantasm and PowerFantasm. Fantasm will assemble 68k assembly language, and PowerFantasm assembles both 68k and PowerPC assembly language.

PowerFantasm assembles for all known 68k and PowerPC processors (including 64 bit instructions), and outputs standard

Macintosh applications. The PowerPC assembler handles all 32 bit "extended" instructions as defined in the architecture.

PowerFantasm runs in two distinct modes. In Stand Alone mode, Fantasm can output code to RAM for testing, or can create a complete double clickable native application on disk. Build mode uses a linker.

Eddie is Lightsofts' integrated editor designed specifically for programming. Eddie is written in 100% machine code (as is Fantasm) for exceptional speed and responsiveness. Eddie has been written from scratch and does not rely on third party code to break the 32k limit.

Fantasm V4.1 requires System 7, a Mac with at least an 020 processor and will run in a minimum of 1 Meg of RAM.
<http://www.tau.it/lightsoft>

New SentinelWizard GUI

Rainbow Technologies, Inc. announced the SentinelWizard GUI (graphical user interface). The SentinelWizard, part of the latest software release for the SentinelSuperPro, guides developers in easily implementing and integrating advanced protection into their SentinelSuperPro key.

SentinelWizard makes implementing SentinelSuperPro software protection easy. SentinelWizard automatically programs their SentinelSuperPro key and generates pseudo code to complete the protection process.

WebSiphon Web-Based Scripting Tool

Purity Software, Inc. announced that it is shipping WebSiphon, its web-based scripting tool for Macintosh computers. WebSiphon is one of the most anticipated new CGI (Common Gateway Interface) products for Macintosh web masters delivering a complete authoring tool for revolutionary sites.

In addition, WebSiphon includes Purity's database server, Verona, a low overhead, flat-file database server created specifically for use on the Internet. Verona is fully multi-threaded allowing multiple concurrent operations among many databases being served on the web. Unlike other current authoring solutions offered to Mac OS web masters, WebSiphon completely integrates the SiphonScript language within the HTML page giving the user the power of a fully featured language. SiphonScript supports variables, conditional statements, powerful looping statements, built-in functions, forms processing, and more. An extensive built-in library offers 79 functions users may choose from to aid in creating their site.

WebSiphon requires an Apple Macintosh computer running System 7.5 or higher, 1 MB RAM for WebSiphon, and a

Macintosh web server that supports "sdoc" CGI AppleEvents. For database functionality, Verona requires an additional 1MB of RAM. WebSiphon has been tested and is fully compatible with the following web servers: Boulevard and Web For One by ResNova; WebSTAR by StarNine Technologies, a division of Quarterdeck; NetPresenz by Peter Lewis; WebCenter by Chris Hawk; and Sonic Systems' forthcoming SonicWeb.

WebSiphon has a MSRP of $495, which includes the flat-file database server Verona (available separately for $195). <http://www.purity.com/>

S-CASE 3.0 for Macintosh

MultiQuest announces the release of S-CASE 3.0 for Macintosh. This new version satisfies the demand for Unified Modeling Language tools brought about by the increasing popularity of this new method.

S-CASE 3.0 enables analysts, designers and engineers to start using this method right away, utilizing its potential to create simpler, cleaner and more expressive models. S-CASE 3.0 also introduces scripted access to its metamodel. Users can now extract model elements using Tcl scripts, and output them in whatever format they wish, such as C++, Java, Smalltalk, SQL or customized reports.

S-CASE 3.0 provides C++ reverse engineering capability. It also allows automatic layout of parsed models for quick visualization and comprehension. Reverse engineered models can be integrated with existing applications or new projects.

S-CASE 3.0 promotes design reuse by introducing the concept of packages. Packages encapsulate all the elements needed for reuse. S-CASE 3.0 for Macintosh lists at $495. (847) 397-9930. <email: info@multiquest.com>

Metrowerks To Support Embedded Development

Metrowerks Inc. announced that it has signed an agreement with Microware Systems Corp. to extend its CodeWarrior development environment to include support for Microware's OS-9 real-time operating system.

Each product in the CodeWarrior for Embedded Systems series will include the CodeWarrior Integrated Development Environment, or IDE, with full-featured GUI tools and C/C++ source-level debugging support, on-line documentation.

In an effort to provide developers with a choice of platforms from which to work, Metrowerks will be offering both Mac OS and Windows 95/NT versions of the software. These product offerings will enable embedded systems developers to efficiently build applications for PDAs, smart phones, Web TV, set-top boxes, navigational systems, car area networks, game machines, and other communications and graphic intensive platforms. <http://www.metrowerks.com>

MT

by Steve Sisak

TIP OF THE MONTH

Avoiding Preloads...

When you open a resource file, all the resources marked 'preload' are automatically loaded. If the file contains a lot of preload resources, HOpenResFile (or OpenResFile, or FSPOpenResFile) can take a long time, or even fail with an error -108 (not enough memory).

If you do not want preload resources to be loaded when you open a resource file, you should set ResLoad to false before opening the file. For example:

```
SetResLoad ( false );
   refNum = HOpenResFile ( vol, dirID, name, fsRdWrPerm
);
   SetResLoad ( true );
   if ( refNum < 0 )
      err = ResError (); // Which error occurred
   else {
         .....          // file was opened correctly
         }
```

- Marshall Clow, Aladdin Systems

Using Labels to Organize Yourself

Until several months ago, I had never thought too much of Labels; never used them, pretty much ignored their existence altogether. It was only when I found myself repeatedly re-installing and/or upgrading my system (and thus needing to replace all of my custom files from the old System Folder), that I found a truly blessed purpose for the Finder's Labels.

The beauty of Labels is that they provide a generic mechanism for the Finder to aide in the user's organization of files. Other customization features like Comments are not so useful in a lot of cases, because there is no intuitive way to quickly use the information to increase efficiency. Thanks to the "View by Label" feature of the Finder, labels can save the day.

A case where I find this especially useful is in the organization of my System Folder. I will describe a system of Labels which I use to ease the task of keeping track of all the system, custom, and 3rd party files which inevitably find their way into my System Folder.

Assign a unique Finder Label to each of the following categories:

1. System: Files or folders which will be completely re-installed by the Installer in the case of an update or clean System Software install.
2. Custom: Files or folders which were installed or instantiated by some action of the user. Anything which you will want to replicate in your new folder but which the Installer will not create.
3. Contains Custom: For folders which contain some but are not entirely filled with custom files.

Now, go to your System Folder and view by Label. Proceed to label every file and folder with one of these labels. If you have some files which don't fit in these categories, you have 4 additional labels to work with.

Now that this initial time consuming task is out of the way, you are ready to reap the benefits of your effort. Periodically, you can use the System 7.5 Find File to keep your folder up to date. Simply do a Find by "Label is None" in the System Folder, and adjust the labels as appropriate.

When it comes time to re-install or update, you can View by Label to quickly arrange your files. Drag all Custom items directly to the new folder. Open and repeat for all "Contains Custom" folders. Ignore all System items, and spend your valuable mental energy only on unlabeled items you may have missed.

This process has taken a lot of the fear and effort out of making drastic changes to my System Folder. If it still sounds like too much work, then my final suggestion will make you smile: Since almost all of the "thinking" is being done by the Finder, the scriptable Finder can be caused to do almost all of this automatically. With an AppleScript that intelligently traverses the System Folder and does the right thing with the right label, you'll almost never need to agonize over them again.

Daniel Jalkut
San Francisco, CA

Send us your tips or we'll install EvenBetterBusError on your machine! On the other hand, we might just pay you ***$25*** *for each tip we use, or* ***$50*** *for Tip of the Month. You can take your award in goods, subscriptions or US$. Make sure any code compiles, and send tips (and where to mail your winnings) to our* ***Tips e-mail address*** *at tips@mactech.com. (See page 2 for our other addresses.)*

(...and we really mean now.)

Think of it as the CNN* of the Programming Community. Loaded with up to the minute news, and constantly updated with the newest developments in our industry.

MacTech NOW™ brings you up to speed on everything you need to know – instantly! And thanks to the new "fast-download" design, you'll get to the information you want in seconds.

Give it a spin! Check it out today and get access to over 500 pages loaded with news, tips, programming secrets, product reviews, and much more. And for those of you looking for some kicks, there's the ever popular "Programmer's Challenge" section where you'll get to bang heads with the best in the community.

Log on to MacTech NOW. Things are happening right now, and you should be aware of them.

http://www.mactech.com

*CNN is a registered trademark of Cable News Networks and bears no affiliation with MacTech Now, MacTech Magazine, or Xplain Corporation.

UNIFORM RESOURCE LOCATORS

By Jim Straus, URLs@mactech.com

Don't hesitate to notify me at URLs@MacTech.com of any sites that you think would be of interest! As always, the full list is maintained on-line at http://www.mactech.com/URLs.html.

Web Watch

To go along with this month's theme, we will look at Internet Tools for the Macintosh. The most popular area of programming for the Internet is writing CGI programs for Web servers. CGI or Common Gateway Interface originally was a specification for how programs were invoked on Unix servers by web servers. However, the term has now come to be used for any programs invoked by web servers to generate information to be returned as a web page. The most common use for CGI programs is in response to Web forms, but they can be used to generate a different web page each time, in response to the user such as a the users location, something on the server such as the time of day, or both. The most popular Web server is WebSTAR, making it the standard for how CGI programs are invoked.

Jon Wiederspan's CGI Tutorial has not been updated in a while, but it is still an extremely valuable tool for learning to program CGI tools for WebSTAR servers. It covers all the basics of Common Gateway Interface programs, creating CGI programs with AppleScript, working with forms, and working with image maps. Also check out some of Jon's other works, and articles. A great spot to visit and look around for lots of good links out to other sites.

Jon Wiederspan's CGI Tutorial
http://www.comvista.com/net/www/lessons/

Grant Neufeld has written a very nice framework for creating CGI programs. It is a multi-threaded C framework so it can be (and has been) used for heavily trafficked sites. Besides supporting the original AppleEvent model, he is supporting the new WSAPI/CFM model. If you want to make professional, robust CGI programs, this is a site to check out.

Grant's CGI Framework
http://arpp.carleton.ca/cgi/framework/

Of course, if there is a way to write a program with AppleScript, there is a way to do it in Frontier. So the Frontier folks have their own site of hints and tools for creating CGI programs using Frontier. If you like Frontier (and there is a lot to like), this is a site for you.

CGIs in Frontier
http://www.scripting.com/apps/webstar.html

A very useful tool for creating CGI AppleScripts is the CGI OSAX. It simplifies parsing all the information that the web server passes to the CGI program and tokenizes the information for easy usage.

CGI OSAX
http://marquis.tiac.net/software/home.html

Now for some more general purpose TCP programming tools, check out Metrowerks site and Eric Behr's sites. These are both useful compilations of tools and information for MacTCP developers. Eric's site is a very complete discussion of TCP/IP on the Macintosh. From how to install it, to how TCP works, to applications and source code.

Macintosh TCP/IP Programmers
http://www.metrowerks.com/tcpip/index.html

MacTCP notes
http://www.math.niu.edu/~behr/docs/mactcp.html

Of course Apple has a couple of site of interesting tools and information for Mac Internet developers. CyberTech has information on Apple's more official projects. This includes e.g., a fast indexer for web sites, NetFinder, and the AppleSearch ACGI, among others. Also, check out Project X, Apple's Netscape plug-in that gives a 3D flyable view of web sites.

CyberTech	http://www.cybertech.apple.com/
Project X	http://mcf.research.apple.com/

Maxis has a site full of tips for Mac Webmasters. If you want to set up your own web site or just see what is involved, this page has enough links to information to satisfy the most curious.

Maxis Webmaster Page
http://www.maxis.co.uk/maxispages/macwebmaster.html

Thanks this month to Eric Behr, Mark Chally, Andy Goldstein, Grant Neufeld, Nermin Pomrcic, Jim Stephenson, Jon Wiederspan, and many others for their contributions for their suggestions and pointers to new and old sites.

Quickies

Internet Related

Cyberdog Pound	http://www.microserve.net/~dhughes/Frontier
Message Boards	http://messages.webdownunder.com/index.html

Other Programmer Resources

Hypercard Heaven	http://members.aol.com/hcheaven/ PowerPlant
Beginner's Page	http://www.netaccess.on.ca/~breakpt/html/powerplant.html

Vendors, Products and Miscellaneous

EvangeList	http://www.evangelist.macaddict.com/
MacClub	http://tivi.eunet.ch/MacClub/
Mark Chally	http://home.earthlink.net/~chally/

It's about business-to-business, home-to-office, artist-to-client, teacher-to-child.

It's about having the knowledge and the know-how to be the best.

It's about making smart choices today and learning about what you need for tomorrow.

It's MACWORLD Expo.
See the best in Macintosh and complimentary tools and technologies. Explore an exhibit floor with everything you've ever wondered about. . . the internet, networking, databases, education, desktop applications, music, design, publishing, illustration, animation, CAD/CAM, programming, entertainment and multimedia. With a complete conference program featuring over 80 sessions, this is your opportunity to interact with leading experts in the industry and get the knowledge and solutions you've been seeking.

MACWORLD Expo/Boston

MACWORLD Expo/San Francisco

See us on the WWW at:
http://www.mha.com/macworldexpo

For more information call
800-645-EXPO x100

Please send me more information on MACWORLD Expo!

❍ **Boston** ❍ **San Francisco**

I am interested in: ❍ Exhibiting ❍ Attending

MT

Name_____

Title _____

Company_____

Address _____

City/State/Zip _____

Phone_____ Fax_____

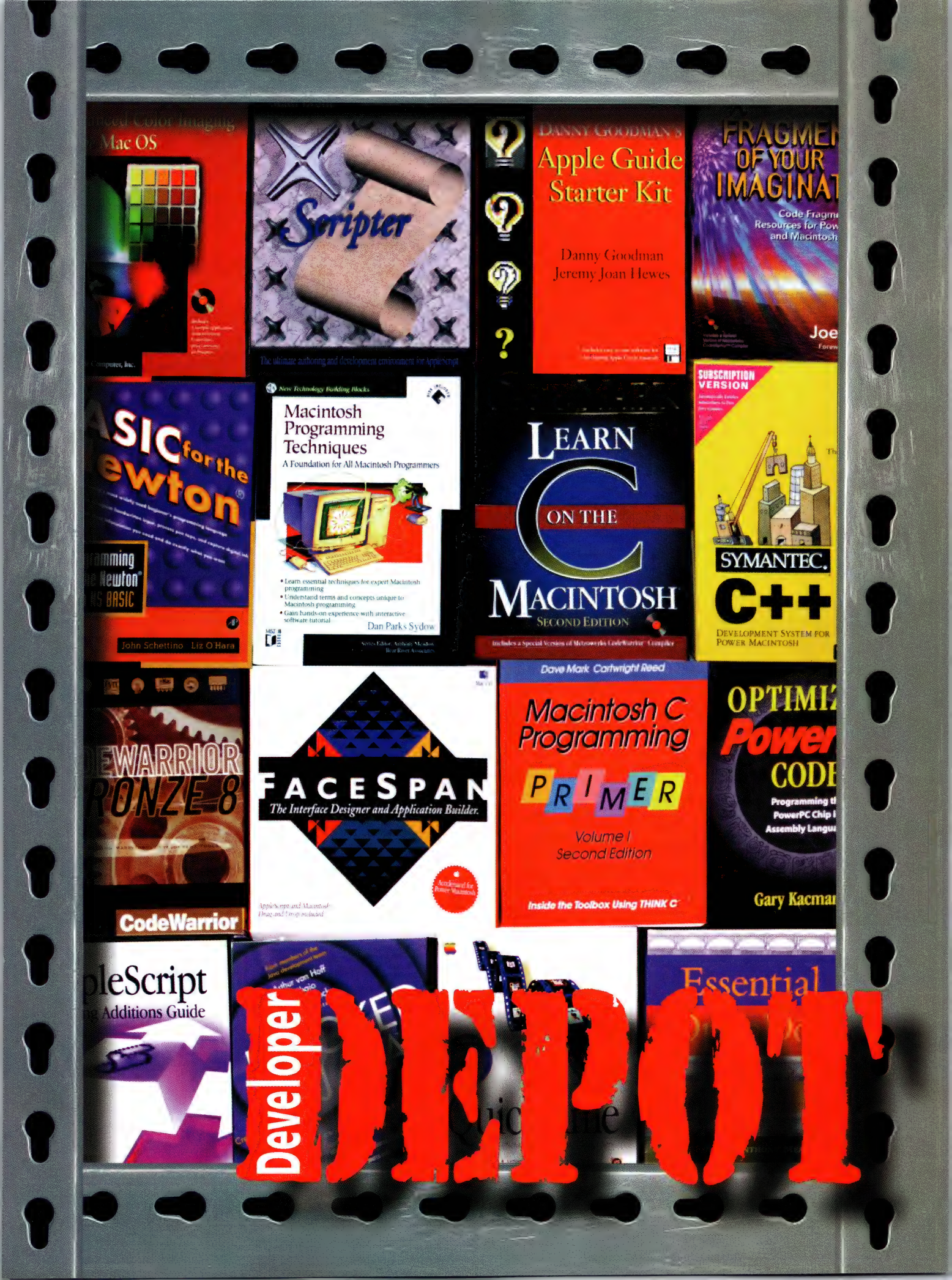
e-Mail_____

Mail to: MHA Event Management, 1400 Providence Hwy., P.O. Box 9103 Norwood MA 02062. Or Fax to: 617-440-0357 Phone: 617-551-9800

ADVERTISER/ PRODUCT LIST

LIST OF ADVERTISERS

LIST OF PRODUCTS


Mac OS
Scripter
The ultimate authoring and development environment for AppleScript
Apple Guide Starter Kit
Danny Goodman
Jeremy Joan Hewes
FRAGMEN
OF YOUR
IMAGINAT
Code Fragme
Resources for Pow
and Macintosh
Joe
SIC for the
ewton
BASIC
John Schettino Liz O'Hara
New Technology Building Blocks
Macintosh Programming Techniques
A Foundation for All Macintosh Programmers
• Learn essential techniques for expert Macintosh programming
• Understand terms and concepts unique to Macintosh programming
• Gain hands-on experience with interactive software tutorial
Dan Parks Sydow
LEARN C ON THE MACINTOSH
SECOND EDITION
SUBSCRIPTION VERSION
SYMANTEC.
C++
DEVELOPMENT SYSTEM FOR POWER MACINTOSH
WARRIOR
RONZE 8
CodeWarrior
FACESPAN
The Interface Designer and Application Builder.
Dave Mark Cartwright Reed
Macintosh C Programming PRIMER
Volume I
Second Edition
Inside the Toolbox Using THINK C
OPTIMI
Power
CODE
Gary Kacmar
leScript
Additions Guide
Essential
Developer
DEPOT!

Dear Developer,

Since March, 1985, **MacTech**™ Magazine (then MacTutor) has been the place for you to find Macintosh Development products – the majors as well as those harder to find. Back then … and today, our focus has always been for the Macintosh developer community to grow and prosper.

But you deserve more than simple listings – and you've asked to make it easier to find and order products while maintaining our renowned customer service and pricing. We've been listening …

We are therefore proud to announce **Developer Depot**™ – a new mail order concept and entity dedicated to you, the Mac OS Developer. What makes Developer Depot so good?

- World renowned customer service
- 24-hour and full accessibility via our continually updated web site (http://www.devdepot.com)
- Satisfaction and best price guaranteed
- A new, easy to remember toll free phone number (**800-MACDEV-1**)
- Great selection (hundreds of products)
- Convenience: you can order in many ways (web site, e-mail, fax, phone)
- A new full color, organized, and expanded catalog – updated monthly

Additionally, we are a Macintosh house using an all Macintosh system. Our knowledgeable staff can answer your questions about products – and if they can't, they'll find the answers for you.

While Developer Depot is completely separate from MacTech Magazine, Developer Depot is the primary source for all MacTech products.

Let us know what you think – we're here to serve you.
Developer Depot: The products you need with the service and prices you deserve.

Neil Ticktin
Chief Executive Officer

Andrea J. Sniderman
Chief Operating Officer

TABLE OF CONTENTS

Developer Depot 30 day Money Back, Price and Satisfaction Guarantee
Developer Depot products are sold with a 30 Day money back guarantee on user satisfaction, lower prices and against defects. If, for any reason, you are not satisfied or find the same product at a lower price within 30 days, please call Customer Service at 800-MACDEV-1 and request a Return Merchandise Authorization (RMA) number to get a full refund or the difference in price (where applicable). You must return undamaged product at your expense, including all its original packaging, documentation and the blank warranty card if applicable. Developer Depot will replace defective product upon receipt of the defective merchandise. Please remember to back up your data before installation of any new hardware, software, or peripherals; we cannot be responsible for any lost data. Policies, item availability, and prices are subject to change without notice. The price in effect when we receive your order will be the price that you are charged. We are not responsible for any typographical errors in this or any other catalog, nor for any misstatements from any vendor. Purchase orders are also accepted but must be in writing, signed, come with a contact person and a telephone number, and mailed to P.O. Box 5200, Westlake Village, CA 91359-5200. Faxed copies and purchase order numbers alone are not acceptable.

Stuff our lawyer made us write.
Developer Depot makes no other warranties. All other warranties, either expressed or implied, including the implied warranty of merchantability and fitness for a particular purpose are disclaimed. Developer Depot shall not be liable for any direct, special, incidental or consequential damages including lost profits, from any delay in delivery, or for any personal injury arising from the use of any product sold through Developer Depot. The limit of direct damages, if any, shall not exceed the purchase price of the product.

For Macintosh Programmers & Developers

MacTech™ MAGAZINE

MacTech Magazine

Subscriptions: US magazine:
$47 for 12 issues (MTYRDM)
Canadian: **$59** for 12 issues (MTYRCM)
International: **$97** for 12 issues (MTYRFM)
Back Issues: **$5** each (subject to availability)

MacTech CD-ROM Volumes 1-11 NEW!

- 1420+ articles, from all 127 issues of MacTech Magazine (1984-1995).
- Improved hypertext, and a new THINK Reference Viewer — for lightning quick access!
- New hyperlinks between articles allow you to jump to other relevant articles.
- 100+ MB of source code — use them in your own applications, with no royalties!
- Full version of THINK Reference™ 2.0. The original online guide to Inside Macintosh, Vols. I-VI.
- 80MB of FrameWorks/SFA archives. The most complete set of FrameWorks archives known.
- Sprocket™! MacTech's Tiny Framework that compiles quickly and supports System 7.5 features.
- The best threads from the Macintosh programmer newsgroups plus thousands of notes, tips, snippets, and gotchas.
- Popular tools that Macintosh programmers use to increase their productivity and much more!

Our Price **$89.00** (SMTCD11) Upgrades: Our Price **$39.00** (SMTCD11U)

NEW!

MacTech Mouse Pad

Slide on this! With an extra-large surface (11" by 10") and a deluxe sleek plastic coating, you'll be zooming across your screen in no time at all. Speed limit not enforced!

Our Price **$8.95** (AMTPAD)

Order Toll-free
800-MACDEV-1
(800-622-3381)

EXCLUSIVES!

FrameWorks Magazine

$8 per back-issue, subject to availability. (MTFWBACK)

FrameWorks Source Code Disk

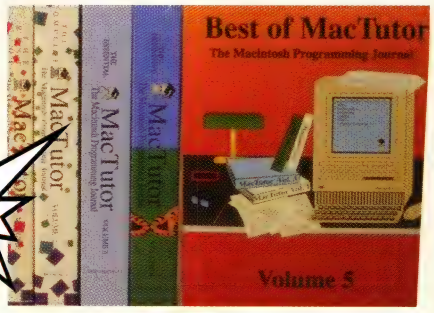
$10 per back issue, subject to availability. (MTFWSC)

MADACON '93 CD-ROM

The highlights of MADACON '93, including Mike Potel on Pink, Bedrock, MacApp, OODLs, and more. Slides, articles, demos, audio, and QuickTime.

List $95.00 Our Price **$9.95** per set (SMADA93).

BLOW OUT!



Best of MacTutor

The Best of MacTutor Collection, Vols.3-5.

List $99.00 Our Price **$9.95** per set (BMTBEST).

CodeWarrior 10 Gold by Metrowerks

- New improved IDE - graphical browser view, global preferences and an improved external editor!
- Full Java toolkit: project manager, linker, editor, class browser, source-level debugger, applet viewer and more!
- Everything you need for industrial-strength programming. Develop for Macintosh, Power Macintosh, Windows 95, Windows NT, Magic Cap and BeOS.
- Award-winning, easy-to-use IDE supports C/C++, Object Pascal and Java.
- Includes online books, extensive reference material and Apple Guide files for easy navigation through tutorials and examples. Two free updates and technical support included with registration.

Our Price **$399.00** (SCWGOLD)

SEE RELATED CATEGORY: **Internet Related**

SEE RELATED PRODUCTS:
• Code Manager • Inside Power Plant • Inside CodeWarrior 9 • C++ Programming with CodeWarrior • PowerMac Programming Starter Kit • Discover Programming for Macintosh • Learn C on the Macintosh • Metrowerks CodeWarrior Programming

CodeWarrior 8 Bronze

- Build applications for the Mac OS on 68K machines.
- Includes C, C++ and Object Pascal compilers, source-level debuggers, object-oriented frameworks (PowerPlant, MacApp), Apple's MPW, complete onlinedocumentation and source code examples for all languages and platforms.
- The IDE software has been localized in eight languages plus English.
- Sold on an annual subscription basis and has three major releases each year.

Our Price **$149.00** (SCWBRON)

CodeWarrior Wear (XL only)

You live it, you breath it... you might as well wear it!

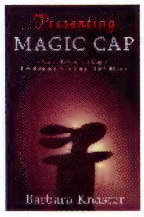
- Black CodeWarrior Sweatshirt: Our Price **$29.95** (ACWSWEAT)
- "Blood, Sweat & Code" black long-sleeve shirt: Our Price **$14.95** (ACWBLOOD)
- Cross Platform white shirt (not pictured): Our Price **$14.95** (ACWXPS)
- Hawaii Five-0 shirt (not pictured): Our Price **$7.95** (ACWHAWAII)
- Arnold at work T-shirt (not pictured): Our Price **$9.95** (ACWARNLD)
- Hat (black or white): Our Price **$14.95** (ACWHAT)
- Winter Hat (not pictured): Our Price **$14.95** (AWINHAT)

CodeWarrior Mouse Pads

Fight your way through the night with these cool-looking pads!

- Arnold: Our Price **$8.95** (ACWPADA)
- Gears (not pictured): Our Price **$9.95** (AMGEAR)

Presenting Magic Cap, A Guide to General Magic's Revolutionary Communicator Software



- Perfect for novices as well as experts who want to take full advantage of Magic Cap.
- Step through its screens, see how it works, and learn ways to use it.
- Describes the power of smart messaging to customize the way you handle e- mail, a name card file that learns how to keep track of your contacts, a datebook that performs like a faithful assistant, and countless other abilities.

List $16.95 Our Price **$15.25** (BPRESMAGIC)

SmalltalkAgents® by Quasar Knowledge Systems

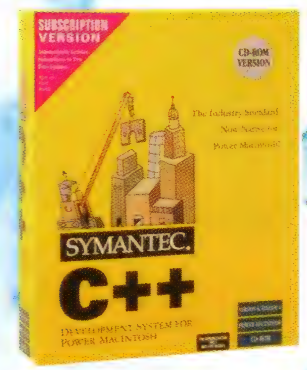
• Features a new generation of the Smalltalk language, QKS Smalltalk™. • Productivity is no longer measured in lines of code, but in project completion time. • Allows "live" direct manipulation of your objects. • Provides a dynamic, interactive and iterative development process. • Provides easy and full access to the features of the Mac OS™ and Mac Toolbox. • Link your non-Smalltalk code resources using our External Code Linking Toolkit™ (ECLT). • A sophisticated database for source code management provides an almost infinite variety of ways to cross-reference, access, view, and manipulate your code and objects. • Includes an Application Delivery Toolkit™ (ADT) that allows you to create royalty-free, stand-alone, double-clickable applications in just a matter of minutes. • Any component built in AO/S will function as an OpenDoc component or container and components from non-AO/S sources can be seamlessly integrated into the AO/S system.

Our Price **$695.00** (SSTA)

SEE RELATED CATEGORY: **Internet Related**

SEE RELATED PRODUCTS: Symantec C++ Programming
SEE RELATED PRODUCTS: Learn C++ on the Macintosh
SEE RELATED PRODUCTS: Programming in Symantec C++
SEE RELATED PRODUCTS: Mastering theTHINK Class Library

Symantec C++ 8.5



- Native for Power Macintosh.
- Develop full-featured Power Macintosh applications quickly and easily!
- Environment includes: a true native Power Mac implementation of the C++ language, MrC/MrC++ compilers for fast Power Mac executable code, Visual Architect for fast, easy GUI generation, a new THINK project management system that supports complex applications, a new editor/browser , a powerful, easy-to-use source code debugger.
- Also includes: The industry-standard THINK Class Library!
- Next two updates free when you send in your registration card and more!

List $499.00 Our Price **$349.00** (SSYMCPP)

Symantec C++ for 68k

- The standard in development languages. • Powerful combination of fully integrated visual tools and the latest in C and C++ compiler technology. • Provides an object-oriented approach to application development. • Write code that is extensible, reliable, and maintainable. Includes: Integrated Environment with full source-code debugging, Incremental Linker which eliminates long link times, THINK Class Library, AppleEvents, Visual Architect™, THINK Inspector, Project Models, Source-Code Control with integration with Apple's SourceServer (included), Support for Scripting, Powerful Standard Libraries which includes I/O Streams, ANSI standard C library, and sample programs. • Full source code is included – Create applications that run on 68K Macs and Power Macs (emulated) These applications can easily be migrated to native Power Mac, by trading up to Symantec C++ for Power Mac.

List $299.00 Our Price **$99.00** (SSYMCPP68K)

Think Pascal Version 4.0

by Symantec Corporation

- Great for professionals and students
- Fully integrated for rapid turnaround time – take advantage of System 7 capabilities
- Supports large projects, enhanced THINK Class Library, System 7 compatibility, superior code generation, and smart linking.
- Includes four Macintosh disks, a user manual, and an object-oriented programming manual.

Our Price **$169.00** (SPASCAL)

MachTen – Power UNIX by Tenon Intersystems

- Dynamically linked shared libraries, memory mapped file access and integrated UNIX and Macintosh development tools.
- BSD 4.4 and conforms to the Federal Information Processing Standard 151-2 (the POSIX FIPS).
- Pre-emptive multitasking for UNIX applications and includes a full featured high-performance TCP/IP protocol stack that supports multi-homing and multi-casting, features not yet available even with Apple's new Open Transport.
- A complete UNIX software development environment with a source-level debugger and C, C++, and Fortran compilers all generating native PPC code.
- Also included is a high-performance X server and complete X11R5 X.

Our Price **$695.00** (SM10PPC)

SEE RELATED CATEGORY: Internet Related

Personal MachTen for 68K Macs

- MachTen UNIX for Macintosh (from Classic on up, including PowerBooks and Duos)
- A Mach-based Berkeley UNIX with pre-emptive multi-tasking
- Extends the Mac OS with UNIX networking and software development tools.
- Built-in internet services include domain name service, POP mail service, internet routing, SLIP & PPP, and Web service.

Our Price **$495.00** (SM10PER) Also Available: MachTen XWindows Our Price **$350.00**

SEE RELATED CATEGORY: Internet Related

Mjølner BETA System

• Software development environment supporting object-oriented programming in the BETA programming language. • Uniquely expressive and orthogonal. • Unifies just about every abstraction mechanism – including class, procedure, function, coroutine, process and exception. • Includes: general block structure, strong typing, whole/part objects. The compiler: binary code generation, automatic garbage collection, separate compilation, interface to C, Pascal, and assembler. • The system: persistent objects, basic libraries with containers classes, platform-independent GUI application frameworks on Unix, Mac and Windows NT, metaprogramming system. • The Mjølner BETA System for Macintosh requires MPW (basic set) 3.2 or later. Package containing compiler, basic libraries, persistent store, GUI framework, and comprehensive documentation. (Other packages are also available).

List $295.00 Our Price **$245.00** (SMJOLNER)

LS Fortran by Fortner Research

• 68K or PowerMac LS FORTRAN comes in two versions: one that generates 68K code (with or without FPU) and one that generates native code for the PowerMac • Source code is compatible between these two compilers • Programs originally developed with the 68K version can boost execution speed by 4 to 6 times by running the same program on a PowerMac • Supports all VAX intrinsic functions and data types, and virtually all VAX extensions • Built-in Debugging • Free Runtime Licenses • Full-featured editor, multiple window capabilities, and a powerful scripting language • Highly optimized, full ANSI standard, 32-bit clean, FORTRAN 77 compiler • Fully compatible with virtual memory under System 7 • Supports a variety of third-party tools to customize any FORTRAN environment • Power Tools include graphing software, subroutine libraries, and user interface tools • Free technical support.

Our Price **$695.00** (SLSFORT)

Fortran 77SDK by Absoft

For Power Macintosh includes a globally optimizing native compiler and linker, native Fx™ multi-language debugger, and Apple's MPW development environment.

• The compiler is a full ANSI/ISO Fortran 77 implementation, and includes all MIL-STD 1753 extensions, Cray/Sun-style POINTER, and several Fortran 90 enhancements • MRWE, Absoft's framework library is included in the MIG graphics library • Supports the native Macintosh PPC toolbox • Includes Absoft's Fx debugger which can debug intermixed FORTRAN 77, C, C++, and PPC assembler • The linker compiler, and debugger all run as native PPC tools, and produce Macintosh PPC executables

Our Price **$699.00** (SF77)

NS BASIC for the Newton

- A fully interactive implementation of the BASIC programming language.
- Runs entirely on the Newton – no host is required.
- Create files, access the built-in soups, and the serial port for input and output.
- Work directly on the Newton, or through a connected Mac/PC and keyboard.

Our Price **$94.99** (SNSBASIC)

LPA MacProlog

- Comprises a Edinburgh syntax Prolog compiler system set in an attractive multi-window development environment with an integrated program editor, graphical call-graph facilities and an interactive source-level debugger.
- Features high-level access to the Macintosh ToolBox for using graphics, dialogs, windows, icons, resources in a simple and versatile way.
- Includes interfaces to C and Pascal code resources.
- Enables the production of double-clickable distributable applications.
- Optional add-ons tools include flex, Prolog++, MacDBI for Oracle and the MacProlog Dialog Editor.

Programmer Edition **$745.00** (SLPAP) Developer Edition (includes the run-time generator and distribution license) **$1500.00** (SLPAD)

MacFortran II V 3.3 by Absoft

MacFortran II is a VAX/VMS compatible full ANSI/ISO Fortran 77 compiler including all MIL-STD 1753 extensions

- Comes with the latest version of MPW, and includes SourceBug, and SoftwareFPU
- Includes Absoft's Macintosh runtime Window Environment (MRWE) application framework, and MIG graphics library
- MacFortran II features improved 68040CPU support and is fully compatible with Power Macintosh under emulation

List $595.00 Our Price **$549.00** (SFORT2)

CodeWarrior Software Development Using PowerPlant by Jan L. Harrington

• C++ programmers will learn to develop object-oriented software applications for the Mac and Power Mac using the PowerPlant environment and the classes that support it. • Covers CodeWarrior 8. • Included CD-ROM contains source code for all the programming examples in the book and Metrowerks CodeWarrior Lite.

List Price: $34.95 Our Price: **$31.45** (BCWSWDEV)

Order Toll-free
800-MACDEV-1
(800-622-3381)

Roaster Subscription™

SEE RELATED CATEGORY: **Dev. Environments**

From Natural Intelligence Inc. Get the most out of Sun's new Java™ Programming Language!

- Write, test and run Java™ applets on the Macintosh in a full-featured Mac development environment
- Features include: project window that includes a finder-like view of packages, Macintosh native compiler, source code editor with powerful search features and intuitive use interface, runtime engine for quick and easy applet testing.
- Requirements : PowerPC, CD-ROM.
- Price includes the current DR 2 release and the next two non-developer releases.

List $399.00 Our Price **$299.00** (SROAST)

Roaster™

From Natural Intelligence Inc.

- Price includes the current DR 2 release and the first non-developer release.

List $179.00 Our Price **$129.00** (SROAST1)

TCP/IP Scripting Addition™

- Award-winning AppleScript scripting addition
- Allows you to write scripts using MacTCP™ commands in AppleScript™.
- Send e-mail or files through a script, check if users are logged on (via Finger), automate FTP, Gopher, NetNews, Telnet, and LPR, verify links in HTML documents, and quickly write many other TCP/IP client-server programs.
- Works with AppleScript 1.0 or later and MacTCP 2.0.4 or later. Compatible with Open Transport™ 1.1.

Our Price **$49.00** (STCP)

SEE RELATED CATEGORY: **Scripting**

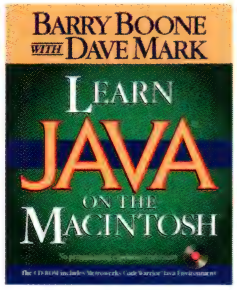
OOFILE HTML Writer

The first OODBMS framework to offer a complete solution for application authors. Replaceable backend database, currently Faircom's c-tree Plus for cross-platform royalty-free power. PowerPlant and other frameworks integrated with edit fields, database browsers and more. AppMaker users, generate complete applications. User-friendly syntax makes it easier to migrate from FoxPro and the non-OO world. Demo's on CodeWarrior and AppMaker CD's. **$895.00** (SOOFCTREE) for a once-off c-tree bundle, or **$1,095.00** (SOOFSUB) 1year subscription. HTML and character-mode report-writer **$195.00**.(SOOFHTML) Full GUI report-writer, including HTML, is **$495.00** (SOOFGUI). Mac Platform Bundle - includes all Mac frameworks, c-tree, and Report-Writer. **$1,495.00** (SOOFBNDL).

SEE RELATED CATEGORY: **Tools, Libs & Utilities**

Learn Java on the Mac

by Barry Boone with Dave Mark.



- Easy-to-follow introduction for beginning programmers and Webmasters.
- Takes you through the core concepts of Java.
- Includes: Object-oriented techniques, unique features such as garbage collection, and basic programming concepts such as working with variables, threads and classes.
- Learn to apply this knowledge to write original Java applets for Web pages.
- Includes CD-ROM

List: $34.95 Our Price: **$31.45** (BLJAVA).

Providing Internet Services via the Mac OS

by Carl Steadman and Jason Snell

- Shows you how to provide Web pages, FTP, Gopher, e-mail, and more with the Mac OS.
- Includes CD-ROM containing all the Mac server software and utilities you need.

List: $34.95

Our Price: **$31.46** (BPROVNET).

Includes CD-ROM

Learn HTML on the Mac

by Dave Mark and David Lawrence.

- Shows you how to use HTML 3.0.
- Included CD-ROM contains Macintosh tools to design stunning multimedia Web pages.

List: $29.95 Our Price: **$26.95** (BLHTML)

Includes CD-ROM

For information: • Voice: 805/494-9797 • Fax: 805/494-9798 • E-mail: marketing@devdepot.com

Developer DEPOT™

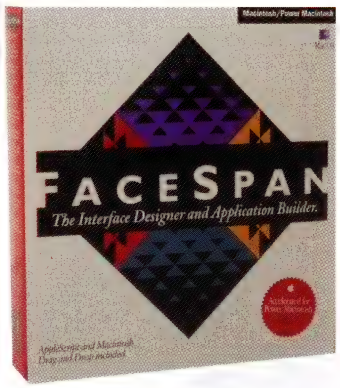
Can't find what you're looking for? Check our Web site: http://www.devdepot.com

1-800-622-3381 • Outside U.S. 805-494-9797 • Fax: 805-494-9798

DataScript

• Make your integrated AppleScript solutions database aware-today. • Takes just six lines of AppleScript to make new or existing scripted solutions database aware, and fetch data from RDBM's such as Oracle, Sybase, DB2, or Informix. • Easy to learn, easy to use, and easy to remember. "Inside DataScript" contains lots of easy to follow scripts to reuse in your own solution. • Very attractive licensing schemes.

List $249.00 Our Price **$229.99** (SWDSCRIPT)



FaceSpan™ v2

• Develop Integrated Software • Make Stand Alone Applications • Create Friendly Interfaces • Develop Quick Prototypes • Print multiple pages with sophisticated layouts • Script essential elements of the FaceSpan application • Play and record sounds as either "snd" resources or as "AIFF" files. • Create miniature or complete applications that will run on either Power PC or 68k computers. • Use precise time measurement for implementing timed behaviors. • Enhanced Save Options • New Properties • Proportionally scale PICT images • Align images in a pictbox • Monitor and respond to low-memory situations • Automate any application • Increased support for Frontier UserTalk!

List $199.00 Our Price **$179.00** (SFACESPAN)

Script Debugger by Late Night Software Ltd.

- A powerful and flexible AppleScript authoring tool – get the most from AppleScript!
- Advanced debugging environment offers single-step script execution with breakpoints.
- Script Debugger dictionary browser features a graphical view of objects provided by scriptable applications.
- Includes Late Night Software Scripting Additions – a collection of more than 70 new AppleScript commands, and Scheduler, a utility that allows you to launch scripts at pre-determined times.

List $129.00 Our Price **$119.00** (SDEBUG)

PreFab Player by PreFab Software, Inc.

- Adds to Your Scripting Language Player adds verbs to AppleScript and to FrontierIs UserTalk. No separate editor or environment. Get the full power of the scripting language at every step. Repeat loops, if/then conditionals, "try" error handling, variables, comments, it's all there.
- Controls the Frontmost Application. Script simply brings an application to the front, then calls player's verbs to perform a task step-by-step. Dialogs will appear, radio buttons will be highlightsed, pop-ups will flash, etc. just as if a user were "playing."
- Balloons Identify User Interface Objects Player adds universal balloon help that shows the name, unique Id, object type and XY location of any dialog or window item. All the information you need, right at your fingertips.

Our Price **$95.00** (SPLAYER)

ScriptWizard™ 1.5

• Combines the power of a professional development environment with the Mac's ease of use. • Compatible with all Apple® Open Scripting Architecture languages, including AppleScript™. • Includes delivering testing and debugging facilities to improve your productivity. • Emphasizes features that speed script development. • Single-step scripts with true statement-level stepping, watch variable values as scripts execute, jump instantly to frequently used places in a script and find and replace specific text.

List $89.00 Our Price **$84.95** (SWIZ)

Scripter 2.0® by Main Event

NEW 2.0!

• Handler debugging and applet simulation: Call and debug any handler in a script at any time. Interactively debug live interapplication messages sent to a script application or AppleScript CGI from an applet, your Web server or FaceSpan 2.x. • Explore an accurate graphical view of the object class hierarchy defined in a scriptable application's vocabulary, to understand the relationship between object types.• Integrated value and object database: ScriptBase (a $59 value) is now included with Scripter. Store your data and media elements (frequently used values, scripts, text, pictures, HTML, headers, file references) and share them between scripts.• Additional editing tools, performance enhancements: Backward search through a script, even more drag-and-drop, and an even more enhanced trace log. Also faster searching through scripts, better speed on PowerBooks, and much more.• Plus all the original tools from Version 1: Power-assisted statement builders, lots of goodies for faster script writing, and the deepest AppleScript debugging tools available.

• Includes a development version of ScriptBase. Inexpensive runtime for ScriptBase licenses are available directly from Main Event.

List $199.00 Our Price **$179.00** (SSCRIPTER)

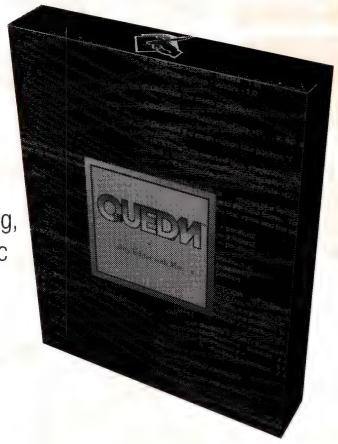
SEE RELATED CATEGORY:
Internet Related

BBEdit 4.0 from Bare Bones Software

- A powerful, easy-to-learn text editor.
- Adds new features for HTML coders, including a spelling checker and HTML tag palette.
- Accelerated for Power Macintosh; dragging supported everywhere; Internet Config aware; PowerTalk aware.
- Integrated support for Symantec's IDE, Metrowerks CodeWarrior, THINK Reference 2.x, MPW ToolServer, and most other environments.
- Many UNIX style tools, including "grep" searches, file comparisons, and sorting. Multi-file search and replace.
- PopUpFuncs feature lets you jump to a function from a menu.

List $119.00 Our price **$94.00** (SBBEDIT)

QUED/M 3.0 by Nisus Software



- The programmer's text editor that defined the industry standard for speed and efficiency.
- Power PC native.
- Features integrated support for Symantec C/C++, Metrowerks CodeWarrior 6, and MPW.
- Supports all the major development environments on the Macintosh.
- Dozens of powerful editing features, including unlimited undo and redo, macro language, scripting, text folding, ten editable/appendable clipboards, markers, displaying text as ASCII codes, dynamic coloring of C/C++ keywords/comments, rectangular and non-contiguous selection.
- Includes Celestin Company's APPRENTICE 4.

List $119.00 Our Price **$69.00** (SQUEDM)

CMaster 2.0 by Jersey Scientific

- Installs into THINK C 5 / 6 / 7 and Symantec C++ for Macintosh and enhances the editor.
- Use its function popup to select a function and CMaster takes you right to it!
- Multiple clipboards and markers, a Function Prototyper, and a GoBack Menu which can take you back to previous editing contexts.
- Almost all features bindable to the keyboard, along over a hundred keyboard-only features like "Add New Automatic Variable." Glossaries, AppleScript and ToolServer support, Macros, and External Tools you create too!

Our Price **$129.95** (SCMASTER)

SEE RELATED PRODUCTS:
Quick Time Starter Kit

Movie Cleaner Pro 1.2 by Terran Interactive

- Compress QuickTime movies
- Powerful and easy-to-use
- Includes: drag and drop batch processing, suspend and resume, adaptive noise reduction, IMA audio compression, high quality crop and resize, A/V fades, gamma correction, de-interlacing and more!
- Automatically suggests the best settings for your application.
- Great for both beginners and experts
- Essential for CD-ROMs or Web sites

Our Price **$189.95** (SMOVIE)

SEE RELATED PRODUCTS:
Quick Time Official Guide

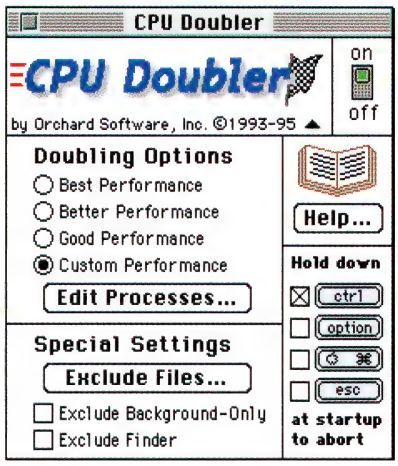
One & 2-Day SHIPPING (call for details)

CodeManager by Metrowerks

SEE RELATED CATEGORY:
Dev. Environments

- Source code control system for the Macintosh.
- Based on and compatible with Microsoft Visual SourceSafe version 4.0.
- Works with any type of binary file including graphic, database, library and executable files.
- Includes support for multi-platform projects, security features, reverse delta versioning of files, comment areas for each revision, project analysis and reporting tools.
- Sold on a subscription basis with two future product updates.
- **Includes a 1 year MacTech subscription.**

Our Price **$399.00** (SCDMGR)



CPU Doubler by Orchard Software

SEE RELATED PRODUCTS:
Cron Manager

- Performance enhancement utility for the Macintosh.
- Increases the speed of your computer by 100%.
- Works on both the PowerPC and 68K Macintosh.
- Manages computer throughput using a proprietary scheduling algorithm.
- Ensure optimal performance and compatibility.

Our Price **$79.95** (SCPU2X)

CLImate by Orchard Software

- Command line interface that lets you communicate with your Macintosh using English commands to create, delete, rename, and move files and folders.
- Start applications, format disks, restart your computer, etc.
- Supplements the Finder.
- Includes a BASIC interpreter that lets you script your Macintosh without AppleScript.
- Includes advanced programming constructs: repeat loops, if/then/else conditionals, subroutine calls, etc.
- Implements wildcard characters, enabling you to work on groups of files.
- Comes bundled with sample programs and full documentation.
- Includes a free copy of Cron Manager – chronological event management utility.

Our price **$59.95** (SCLIMATE)

FILE genie PRO

File Genie Pro by Duet Development

- File Genie Pro lets you search ALL your developer CD-ROMs at once!
- Every Dev CD, OS SDK, CodeWarrior CD, E.T.O., and Bookmark CD you insert is cataloged automatically. A typical Dev CD with 10,000 files is cataloged in under 10 seconds.
- You can also catalog opticals, SyQuests, and floppies.
- File Genie Pro quickly finds any file on any disk by searching hard disks, network servers, and catalogs. Open, show, print, and more. View text and graphics files without opening other applications.
- When acting on an ejected disk file, File Genie Pro tells you which disk to insert, then continues automatically.

Our Price **$89.00** (SGENIE)

Order Toll-free
800-MACDEV-1
(800-622-3381)

Guide Composer™ 1.2 by StepUp Software NEW!

SEE RELATED PRODUCTS: Real World AppleGuide
SEE RELATED PRODUCTS: AppleGuide Complete
SEE RELATED PRODUCTS: Danny Goodman's AppleGuide Starter Kit

- Create powerful Apple Guide help systems for any new or existing Macintosh application.
- Great for commercial developers, shareware developers, in-house developers, and consultants.
- Provides a WYSIWYG development environment: Guide content is developed in Guide windows.
- Design topics, phrases, and panels in the same format as the user will use them. Features are WYSIWYG interface, Topics, phrases, and hierarchical phrases, Coach marks, Fully-Integrated with Apple's Guide Maker (distributed with Guide Composer), compiles scripts automatically, PICTs in Panels, Generated Guide scripts are modifiable.
- Compiled files are 100% AppleGuide-compatible and royalty-free. Easy-to-use.

STEPUP SOFTWARE

New features include:

- Add up to six radio buttons on an Apple Guide panel, each of which can branch to another sequence or topic
- Add up to six checkboxes on an Apple Guide panel, each of which can insert another sequence or topic into the current topic and much more!
- FREE Update to all registered Guide Composer users. Demo is available at http://www.guideworks.com/.

Our Price **$99.00** (SGCOMP)

ScriptGen Pro by StepUp Software

- Installer script generator which requires no programming or knowledge of Rez.
- Supports StepUp's InstallerPack, StuffIt decompression, Compact Pro decompression, custom packages, splash screens, network installs, and resource installation.

Our price **$169.00** (SSCRPTGEN)

Step-Up Installer Pack by StepUp Software

- Package of several Installer "atoms" that let developers incorporate graphics, sounds, file compression and custom folder icons into installation scripts.
- Compression formats supported are Compact Pro & Diamond.
- Each atom also available separately.
- Compression requires additional licensing.

Our price **$219.00** (SINSTALL)

CronManager by Orchard Software

SEE RELATED PRODUCTS: CPU Doubler

- Implements the UNIX Cron facility.
- Open any Macintosh file on a given date and time.
- Simple interface.
- Works with any Macintosh file.
- Cron Manager bundled with CLImate,

Our price **$26.95** (SCRONMGR)

Tenon Ported Application CD

- Contains hundreds of applications that have been pre-compiled to run on Personal or Professional MachTen 68k systems, such as Unix, Perl, and TCL/TK
- Includes many internet servers such as Gopher, WAIS Ported Application CD Vol. 1 (for 68K Macs).

Our price **$49.95** (SPORTED)

Rosanne

SEE RELATED CATEGORY: Scripting

- A collection of utilities which offer the user complete control over raw data.
- Sort files, extract selected records, summarize frequency counts, create sample files, perform matching on multiple files, and reformat data to new specifications, all on the desktop, and even on files of a million records or more.
- Supports AppleScript™, enabling the user to link several actions together to complete an entire process.
- Recordable – users may perform a series of actions, and see them translated directly into AppleScript commands.
- Supports multi-tasking and background processing.
- Rosanne Utilities: Copy, Format, Select, Recode, Sort, Match, and Aggregate

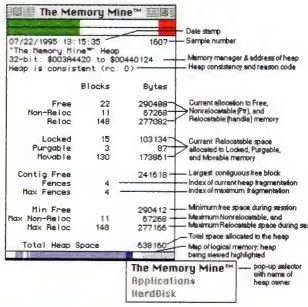
Our Price **$595.00** (SROSANNE)

Memory Mine by Adianta

SEE RELATED CATEGORY: Dev. Environments

- Monitor heaps, identify problems such as memory leaks, and stress test applications
- Active status of memory in a heap is sampled on the fly: allocation in non-relocatable (Ptr), relocatable (Handle) and free space is shown, as are heap corruption, fragmentation, and more
- Allocate, Purge, Compact, and Zap memory lets users stress test all or part of a program
- Works on Macintoshes with 68020 or later and System 7.0 or later.

List $99.00 Our Price **$94.99** (SMEMMINE)



QC by Onyx Technology. High performance runtime stress testing for applications.

- Tests include heap checks, purges, scrambles, handle/pointer validation, dispose/release checks, write to zero, de-reference zero as well as other tests like free memory invalidation and block bounds checking.
- Extremely user friendly – ideal for non-programmer testers.
- Also available in Japanese.

List $99.00 Our price **$94.99** (SQC)

SEE RELATED CATEGORY: Dev. Environments

Last Resort Programmers Edition

- Records every keystroke, command key and mouse event (in local coordinates) to a file on your hard disk
- Great for program testing & debugging, and for technical support and help desks
- Last Resort keystroke files and recovers what you typed – great for data losses caused by power failure, crashes, or accidental deleting.

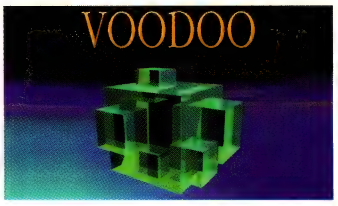
Our price **$74.95** (BLSTRSRT)

Spyer by InCider

- Easy to use tool that records all actions (including mouse movement) you perform on a Macintosh computer and then replays them at your preferred speed.
- Recorded data can be saved in files for future use.
- Works as a background process with any Macintosh application and is triggered by user defined Hot Keys.
- Enables the "Continuous Redo" utility and is especially useful for software testing and demonstration.

Our price **$39.00** (SSPY)

VOODOO



- Version control tool for the simple and clear management of projects in which files are created in numerous versions (variants and revisions).
- Allows both variant and revision control, and it manages not only variants and revisions of single files, but of a whole software project (multi files, multi users, multi variants, access rights, etc.)
- Graphical user interface and is not only suitable for mere source code control but can handle all different kinds of files with amazing compression rates: typical size of delta between arbitrary files 5%
- Please note special prices for multiple copies:

Single license **$229.00** (SVOODO01); 2 pack **$359.00** (SVOODO02); 5 pack **$799.00** (SVOODO05); 10 pack **$1369.00** (SVOODO10); 20 pack **$2399.00** (SVOODO020)
Additional pricing available on request.

SoftPolish CD-ROM by Bare Bones Software

- The essential tool for software quality assurance on the Macintosh
- Helps you identify inconsistencies with Apple's user interface guidelines, misspelled words, missing resources, and other mistakes
- Provides tools to put the finishing touches on software distribution packages prior to release
- Works independently of any programming language or environment
- Ideal for sanity checking software throughout the development process.

List price: $99 Our price: **$89** (SSOFTPOL)

LJ Profiler by Lars Jordebo Datakonsult

- Supports profiling of C++ 68K and PowerPC applications compiled with CodeWarrior, CFront or Symantec C++.
- Based on active profiling, i.e. profiling code called at function enter and exit, the browser application lets you follow call chain timings in hierarchical views or separate windows.
- Collect, organize, compare and save profiling data from different versions of your application into a project. Scriptable and recordable with full access to most internal data structures.
- Optional remote profiling and tracking of segment and stack usage. Full source code to what you link into your application.

Our price **$295.00** (SLJPROF)

PowerTap™

- Accelerates software by tapping into multiple processors.
- Easy API – submit tasks and retrieve results!
- Full error recovery system plus scheduling algorithms for optimal assignments and fastest possible execution.
- Compatible with all Macintosh hardware, software and major compilers.

PowerTap/2, 2 remotes edition List $1,200.00 Our Price **$1,095.00** (SPOWTAP2)
PowerTap/5, 5 remotes edition List $1,900.00 Our Price **$1,795.00** (SPOWTAP5)
PowerTap/n, unlimited edition List $2,700.00 Our Price **$2,495.00** (SPOWTAPN)

One & 2-Day SHIPPING (call for details)

StoneTable

- Replaces all functions found in list manager
- Variable size columns/rows; different font, size, style, forecolor, backcolor per cell; sort, resize, move, copy, hide columns/rows; edit cells/titles in place; titles for columns/rows; multiple lines per cell; grid line pattern/color; greater than 32k data per table; up to 32k text per cell; support for balloon help and binary cell data. Versions for Think C, Think Pascal, MPW C, MPW Pascal, CodeWarrior 6 C.

StoneTable Extra

- Drag selected cells within table or to other tables.
- Add rows as part of drag; popup menus or check boxes in cells; variable width grid lines; move/drag/resize table in window; clipboard operations on multiple cells.
- Requires StoneTable.

This product can now be ordered in 3 different packages.

68K StoneTable

This includes StoneTable, StoneTableExtra for Think Pascal, Think C, MPW C, MPW Pascal, CodeWarrior C, CodeWarrior Pascal

Our price **$49.00** (SSTABLEX)

PPC StoneTable

This includes StoneTable PPC, StoneTableExtra PPC for Think C, MPW C, MPW Pascal, CodeWarrior C, CodeWarrior Pascal

Our price **$99.00** (SSTNXPPC)

68K/PPC StoneTable

Includes both packages

Our price **$325** (SSTONEFAT)

PictureCDEF 1.3 by Paradigm Software

- Professional-level CDEF for creating custom graphical buttons (8-64 pixels) – used in products by Adobe, ProVue, STF Technologies and others!
- Multi-monitor and bit-depth sensitive.
- The button graphic (cicn, ResEdit) can be changed at runtime and even animated with a call-back routine.
- Create distinct buttons in seven variations: MultiState, PushButton, FlexiButton, ToggleButton, ChkButton, PushPictButton and TogglePictButton.
- Manual, sample code and MacApp 3.0 support included.

Full source code: **$95.00** (SPCDEF) Object code: **$45.00** (SPICTOBJ)

SpellsWell 7 1.0.4

- Award-winning, comprehensive, practical spelling checker that works in batch mode or within applications that incorporate the Apple Events Word Services protocol (e.g., Eudora, WordPerfect, Communicate!, and Fair Witness).
- Checks for spelling errors as well as common typos like capitalization errors, spaces before punctuation, double word errors, abbreviation errors, a/an before vowel/consonant, etc.
- MacTech orders include developer kit with Writeswell Jr., a sample AppleEvents Word Services word-processor and its source code.

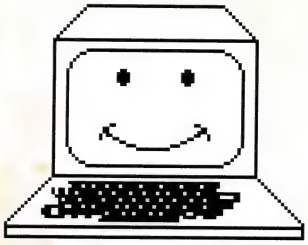
Our price **$74.95** (SSPELL)

MacWireFrame by Amplified Intelligence

- Create your own virtual reality application with MacWireFrame, a virtual reality application framework.
- Includes a complete library of object oriented graphics routines.
- Easy to understand application frame work, plus an example application program that lets you start solid modeling right away.
- Complete with fully documented source code.
- Guaranteed $49.99 upgrade to the soon to be released, scriptable, MacWireFrame 5.0.

List $299.00 Our Price **$75.00** (SFRAM)

Order Toll-free
800-MACDEV-1
(800-622-3381)



B-Tree HELPER™ 2.2

- Inexpensive database engine for Macintosh programmers in C source code.
- Uses contiguous fixed length blocks.
- Expands the file as necessary and contracts files when possible.
- Inserts and deletes keys in one or more B-Trees.
- Finds keys equal to, less than, or greater than a given value in a few hundredths of a second.
- Finds lists of records whose keys are equal to, less than, or greater than a given value or are in a range of values.

Our Price **$150.00** (SBTREE)

Q3S/3dPane/SmartPane

ViviStar CONSULTING

- Full featured 3d graphics library.
- Polyhedra; Gouraud shading; stereoscopic projections; pipeline access; animation and model interaction support; a "triad mouse" to map 2d mouse movement to 3d; and all the regular 3D features.
- 3dPane provides integration with the TCL and provides a view orientation controller. SmartPane provides TCL offscreen image buffering, flicker free animation, and QuickTime movie recording. SmartPane functions in 3d or 2d scenarios.
- All work with C++ compilers or ThinkC 6 and compile to PowerPC or 68K target machines.

Our Price **$192.00** (SQ3)

dtF

- True relational database system for Apple Macintosh computers.
- Provides a powerful choice for developers who want to create database centered applications with no performance trade-offs.
- Features SQL, full transaction control, error recovery, single user, client server architecture and multi-platform support including DOS, Windows, OS/2 and UNIX.
- The C/C++ API is identical and fully portable across all supported platforms.
- Third-party vendors supporting dtF will be able to offer a variety of advanced features and benefits to their customers royalty free.
- Tools are included for importing, exporting, creating and managing databases and users.
- Supported development environments include: Symantec, MPW, Metrowerks and more. Mac/SDK

List $695.00 Our Price **$679.00** (SDTF)

AppMaker

- Develop the user interface for a Macintosh application using the original interface builder.
- Just point and click to design your application.
- Creates resources and generates excellent source code.
- Supports most development environments including Metrowerks, Symantec, or MPW;
  C, C++, or Pascal; procedural or object-oriented, using PowerPlant, TCL, or MacApp.
- The generated code uses the Universal Headers to provide PowerMac compatibility.
- Great tool for beginners to learn object-oriented and Macintosh Toolbox programming techniques.
- Includes one-year subscription on CD.

List $299.00 Our Price **$279.99** (SAPPMAKE)

Includes CD-ROM!

NeoAccess™

- Full-featured object database engine for use in Macintosh, Windows, Unix and DOS based C++ applications.
- Extended binary trees and binary search algorithms tuned for short access times; dynamically combined, collapsed, and compressed indices; object caching for instant access to previously used objects.
- Build fast, powerful applications in record time!

Our price **$749.00** (SNEO)

Lowest price guaranteed. Feel Good about your purchase! 30-day return policy.

Developer DEPOT™

Complete info on these products and hundreds more! http://www.devdepot.com

Web site: http://www.devdepot.com • E-mail: orders@devdepot.com

Programmer's Toolbox Assistant CD-ROM

Instant electronic access to Inside Macintosh essentials.

- Get quick access to reference pages for over 4,000 Toolbox calls in your system software from their development environment.
- Essential information for Macintosh software developers.
- Hypertext links allow programmers to view related topics easily.
- The ultimate electronic reference tool for Macintosh programmers.

List $99.95 Our Price **$89.95** (STBASST)

Inside Macintosh®: Overview by Apple Computer, Inc.

- An overview of Macintosh programming fundamentals and a road map to the New Inside Macintosh library.
- Covers various programming tools and languages, compatibility guidelines and considerations for worldwide development. 176 pages.
- Great for beginners!

List $22.95 Our Price **$12.48** (BIMOVER)

Inside Macintosh®: Macintosh Toolbox Essentials

by Apple Computer, Inc.

- Covers the heart of the Macintosh. The toolbox enables programmers to create applications consistent with the Macintosh "look and feel".
- Describes Toolbox routines and shows how to implement essential user interface elements, such as menus, windows, scroll bars, icons and dialog boxes. 880 pages.

List $34.95 Our Price **$19.98** (BIMTBOX)

Inside Macintosh®: More Macintosh Toolbox by Apple Computer, Inc.

- Managers discussed include Help, List, Resource, Scrap and Sound.
- Covers other Macintosh features such as how to support copy and paste, provide Balloon Help, play and record sound and create control panels.

List $34.65 Our Price **$17.48** (BIMMAC)

Inside Macintosh®: Files by Apple Computer, Inc.

- Describes the parts of the operating system that allow you to manage files.
- Explains how your application can handle the commands typically found in a File menu.
- Provides a reference to the File and Alias Managers, the Disk Initialization and Standard File Packages. 510 pgs.

List $29.95 Our Price **$14.98** (BIMFIL)

Inside Macintosh®: Operating System Utilities by Apple Computer, Inc.

- Describes parts of the Macintosh Operating System that allow you to manage various low-level aspects of the operating system.
- Shows how to get information about the operating system, manage operating system queues, handle dates and times, control the settings of the parameter RAM, manipulate the trap dispatch table, and receive and respond to low-level system errors.

List $26.05 Our Price **$14.48** (BIMOPSU)

Inside Macintosh®: Processes by Apple Computer, Inc.

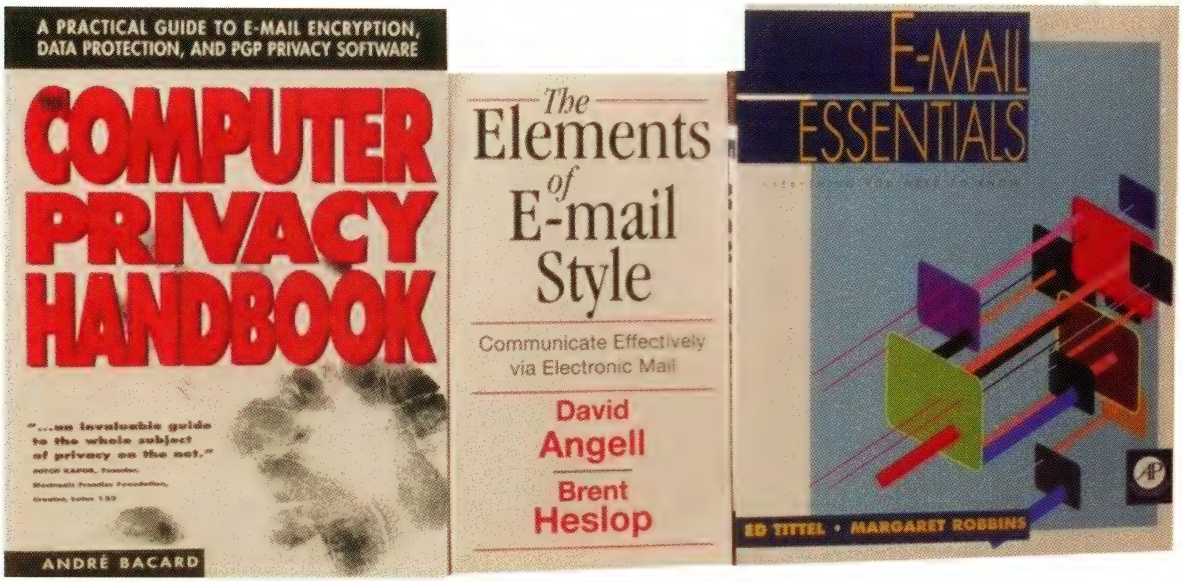
- Describes the parts of the Macintosh operating system that allow you to control the execution of processes and interrupt tasks.
- Shows how you can use the Process Manager to get information about processes loaded in memory.
- A reference for the Vertical Retrace, Time, Notification, Deferred Task, and Shutdown Managers. 208 pages.

List $22.95 Our Price **$11.48** (BIMPROC)

Inside Macintosh®: Memory by Apple Computer, Inc.

- Describes the parts of the Macintosh operating system that allow you to manage memory.
- Provides detailed strategies for allocating and releasing memory, avoiding low-memory situations, reference to the Memory Manager, the Virtual Memory Manager, and memory-related utilities. 296 pages.

List $24.95 Our Price **$12.48** (BIMMEM)



The Computer Privacy Handbook

- A practical guide to e-mail encryption, data protection, and PGP privacy software.

List $24.95 Our Price **$22.45** (BPRIV)

The Elements of E-Mail Style

by Brent Heslop and David Angell

- Write solid, effective E-Mail and avoid common pitfalls.

List $14.95 Our Price **$13.45** (BEMAIL)

E-Mail Essentials

by Ed Tittel & Margaret Robbins

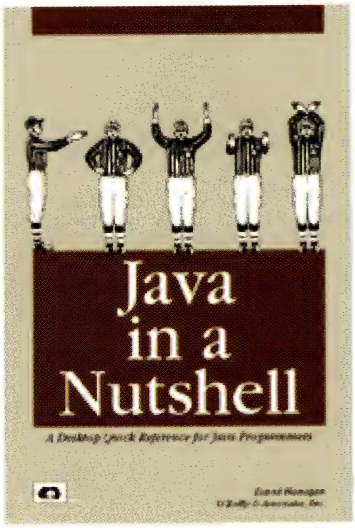
- A hands-on guide to the basics of e-mail.

List $24.95 Our Price **$22.45** (BEMAILE)

Order Toll-free
800-MACDEV-1
(800-622-3381)

Cyberpunk Handbook,

List $9.95 Our Price **$8.95** (BCYBPUNK)



Java in a Nutshell

- A complete quick reference guide to Java, the hot new programming language from SunMicrosystems.
- Contains descriptions of all of the classes in the Java 1.0 API, with a definitive listing of all methods and variables.
- Also contains an accelerated introduction to Java for C and C++ programmers who want to learn the language fast.

List $14.95 Our price **$13.45** (BJAVANUT)

The Instant Internet Guide

List $14.95 **$13.45** (BINSTANT)

Lowest price guaranteed. Feel Good about your purchase! 30-day return policy.

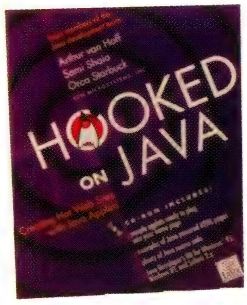
Planning and Managing Websites

Includes CD-ROM!

- The definitive guide to setting up and running a Web site on the Macintosh.
- Learn everything you need to know about using WebSTAR, the best known HTTP server software and its shareware predecessor MacHTTP.
- Write CGI applications for your server – in AppleScript and in C.
- CD includes a special version of WebSTAR, plus tons of useful software.

List $39.95 Our Price **35.96** (BPLANWEB)

SEE RELATED CATEGORY: Internet Related



Hooked on Java

- Written by the Java development team at Sun.
- An introduction to using applets, for Web administrators, designers, and developers.
- Demonstrates how to use applets in your own pages.
- Includes a concise introduction to the Java language, and a CD with tools.

List $29.95 Our Price **$26.95** (BHJAVA)

Includes CD-ROM!

Teach Yourself Java for Macintosh in 21 Days

- Add interactivity and multimedia to Web pages!
- Astep-by-step guide to make your Web site come alive.
- Learn the basics of programming Java applets and the concepts behind the Java language.
- Includes CD-ROM with a limited version of Roaster, the first commercial, integrated applet development environment for Java for the Macintosh!

List $40.00 Our price **$36.00** (BJAVAMAC)

JavaScript for the Macintosh

- Allows non-programmers to take advantage of the power of Netscape Navigator.
- Expand the capabilities of your Web page, without having to understand C or C++.
- CD-ROM contains "Wizlets" that allows you to easily create your own JavaScripts
- Takes you step-by-step through programming cross-platform JavaScripts
- Details how to create JavaScripts for JavaScript-aware Web browsers

List $45.00 Our price **$40.50** (BJAVASCRPTJ)

Includes CD-ROM!

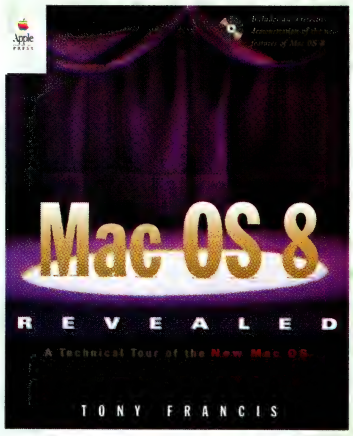
Mac OS 8 Revealed by Tony Francis

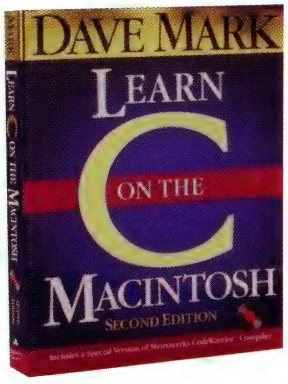
- The first authoritative look at this exciting new operating system.
- A must for Mac developers who want to make their software compatible with Mac OS 8.
- essential for system administrators who plan to upgrade their system.
- Included CD-ROM contains demos of new Mac OS 8 features.

List: $34.95 Our Price: **$31.45** (BMACOS8R)

Includes CD-ROM!

One & 2-Day SHIPPING (call for details)





Learn C on The Macintosh Second Edition By Dave Mark

- New revised edition.
- Easy-to-understand – everything you need to start programming!
- Updated and enhanced exercises that lead you step by step. You'll learn function, variables, pointers datatypes, data structures, file input and output and more!
- Includes CD-ROM with Metrowerks CodeWarrior™ Lite – the hottest Macintosh programming environment (including a PowerPC version).

List $34.95 Our Price **$31.45** (BLEARNC2)

SEE RELATED CATEGORY: **Dev. Environments**

Discover Programming for Macintosh

- Includes full working version of CodeWarrior along with three online tutorial books and Dave Mark's "Learn C on the Macintosh" converted to AppleGuides.
- Includes C, C++ and Object Pascal compilers for generating 68K Macintosh code, source-level debuggers, object-oriented frameworks (PowerPlant, MacApp), Apple's MPW, complete online documentation and source code examples for all languages and platforms.
- The IDE software has been localized in eight languages plus English. This product is not sold as a subscription.
- **Includes a 3 month subscription to MacTech Magazine.**

Our Price **$79.00** (SCWDISCMA)

SEE RELATED CATEGORY: **Dev. Environments**

CGI™ by Jeffrey Dwight

- CD-ROM contains all the code and tools used in the book, and additional tools and examples, as well as related chapters from Special Edition Using CGI and Special Edition JavaScript
- Includes end of the chapter review questions and exercises to reinforce the learning process
- Covers basic CGI applications, as well as advanced topics like server administration issues and database connectivity

List $$34.99 Our Price **$34.99** (BSGIBE)

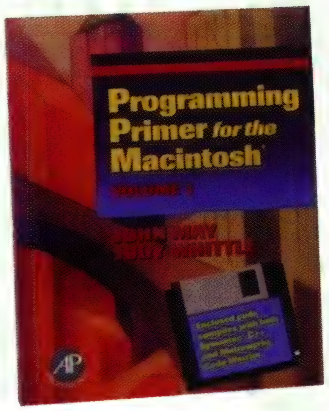
Perl Quick Reference™ by Michael O. Foghlu

- Commands are sorted by task, class packet, platform or hardware compatability all in alphabetical order
- Includes jump tables to guide reader to specific pages in the reference
- Designed in a larger trim size than traditional Quick References-and with a lay-flat binding

List $$19.99 Our Price **$17.99** (BPERLREF)

Macintosh C Programming Primer Volume I

Second Edition, Inside the Toolbox Using THINK C by Dave Mark and Cartwright Reed



- Updated new edition of the Macintosh programming best seller.
- System 7, new versions of THINK C and ResEdit.
- Learn how to use the resources, Macintosh Toolbox and interface to create stand-alone applications.
- 672 pages.

List $26.95 Our Price **$24.25** (BCPRIM1)

Macintosh C Programming Primer Volume II

Mastering the Toolbox Using THINK C by Dave Mark.

- Covers advanced topics such as: Color QuickDraw, THINK Class Library, TextEdit, and the Memory Manager: 528 pgs.

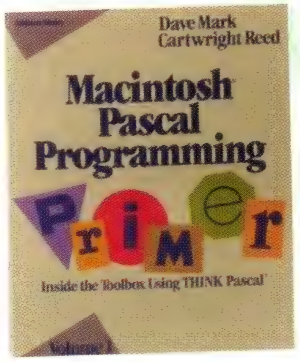
List $26.95 Our Price **$24.25** (BCPRIM2)

Learn C++ on the Macintosh by Dave Mark.

SEE RELATED CATEGORY: **Dev. Environments**

- Basic syntax of C++ and object programming.
- Learn how to write, edit, and compile your first C++ programs.
- Features key C++ concepts such as derived classes, operator overloading, iostream functions and more.
- Includes a special version of Symantec C++ for Macintosh. Book/disk package with 3.5" 800K Macintosh disk. 400 pages.

List $36.95 Our Price **$33.26** (BLRNCPP)

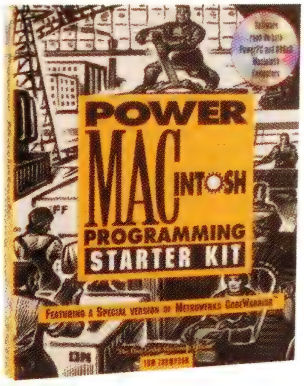


Macintosh Pascal Programming Primer Volume I

Inside the Toolbox Using THINK Pascal by Dave Mark and Cartwright Reed.

This tutorial shows programmers new to the Macintosh how to use the Toolbox, resources, and the Macintosh interface to create stand-alone applications with Symantec's THINK Pascal. 544 pages.

List $26.95 Our Price **$24.25** (BPASCPRI)



Power Macintosh Programming Starter Kit

by Tom Thompson.

SEE RELATED CATEGORY: **Dev. Environments**

- Enter the world of the PowerPC chips.
- Get the scoop on the microprocessors, the RISC architecture, and how to write native code and emulation operations to create software for the Macintosh PowerPC.
- CD-ROM includes a unique compiler for writing code easily.

Includes CD-ROM!

List $39.95 Our Price **$35.10** (BPPCSTART)

Macintosh Programming Secrets 2nd edition

By Scott Knaster, and Keith Rollin

- Macintosh Programming Secrets is divided in two parts.

Part 1, "Concepts and Ideas", discusses the evolution of the Macintosh and the standards, customs, and software that shape the system as well as the Macintosh user interface.

Part 2 "Technical Adventures", presents the skeleton of an application, and then builds upon that framework to describe how to: • Create fancy dialogue boxes • Utilize the new 32 bit QuickDraw developments • Track the mouse with "marching ants" • Manage multiple windows with the Window manager • Copy files within a program • Install the worlds strangest spinning cursor.

List $31.95 Our Price **$28.76** (BPSECRET)

Order Toll-free
800-MACDEV-1
(800-622-3381)

Cyberdog Programmers Kit by Apple Computers, inc.

- Apple's official reference.
- A must for developers who want to make customizable Internet access available to all Mac users.
- Included CD-ROM contains all the tools needed to create Cyberdog-aware components.

List: $34.50 Our Price: **$31.05** (BCYBERDOG)

Programming with AppleTalk by Michael Pierce

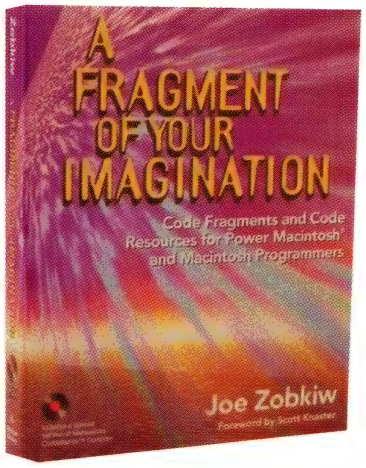
- Programming with AppleTalk is the hands-on guide to understanding and working with AppleTalk. Topics covered include:
- How to create applications and system extensions that run with AppleTalk.
- AppleTalk protocols and the protocol stack, transport media, the Preferred AppleTalk Intefrace, and the storage management.
- Numerous working code examples walk you through using RDEV, INIT, NBP, ATP, and ADSP You will also learn the use of: Synchronous, and asynchronous calls, How to avoid heap fragmentation, And how to configure a Chooser Interface.

List $24.95 Our Price **$22.45** (BPROAT)

A Fragment of Your Imagination by Joe Zobkiw

- Packed with useful code fragments for the Macintosh and Power Macintosh.
- Hard to find information about techniques used to structure and build fat, safe fat, and accelerated code resources.
- All code is reusable and is provided on the disc, along with Metrowerks Code Warrior Lite. Book/CD-ROM, 528 pages.

List $39.95 Our Price **$35.96** (BFRAG)



Java Language API SuperBible

by Daniel Groner, Todd Sundtsed, Casey Hopson, Harish Prabandham

- Covers Java 1.1
- Hundreds of concrete source code examples provided
- Sample projects introduced and assembled in each chapter

List $59.99 Our price **$53.99** (BJLAS)

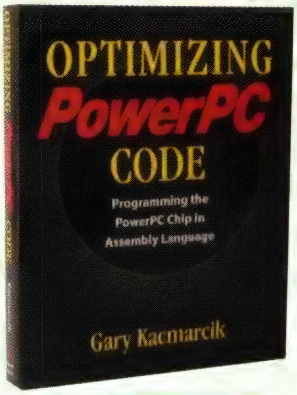
Infini-D Revealed

by Brendan Donahoe & Adam Lavine

- CD-ROM includes a demo version of Infini-D 3.1, files for working through the tutorials, and a gallery of spectacular 3D images
- Shows how to use the new animation features to bring images to life
- Details Infini-D's new and improved color-coded interface

List $45.00 Our price **$40.50** (BINFDREV)

Optimizing PowerPC Code: Programming the PowerPC in Assembly Language



- Take full advantage of the potential of the PowerPC by mastering the Assembly Language techniques.
- Learn to produce faster more robust software!

List $39.95 Our Price **$35.96** (BOPTPPC)

Inside Macintosh®: PowerPC Numerics

by Apple Computer, Inc.

- Describes the floating-point numerics environment provided with the first release of PowerPC processor-based Macintosh computers.
- Provides a description of the IEEE standard 754 for binary floating-point arithmetic., and how RISC Numerics compiles with it.
- Shows programmers how to create floating-point values and how to perform operations on floating-point values in high-level languages such as C and in PowerPC assembly language.

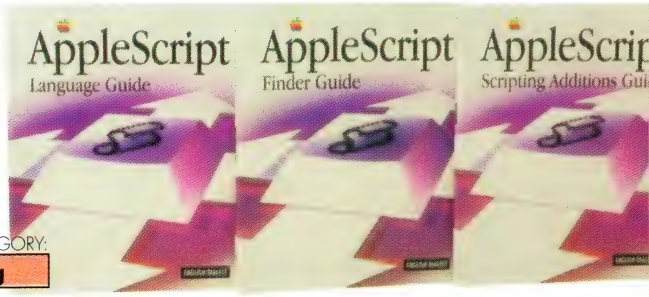
List $28.95 Our Price **$14.48** (BIMPPCNUM)

Inside Macintosh®: PowerPC System Software by Apple Computer, Inc.

- Describes the new process execution environment and system software services provided with the first version of the system software for Macintosh on PowerPC computers.
- Contains information to write applications that can run on the PowerPC.
- Shows how to make your software compatible with the new run-time environment provided on PowerPC-based Macintosh computers. It also provides a complete technical reference for the Mixed Mode Manager, the Code Fragment Manager, and the Exception Manager.

List $24.95 Our Price **$12.48** (BIMPPCSYS)

AppleScript Finder Guide, English Dialect by Apple Computer, Inc.



- Provides definitions for Finder object classes and commands.
- Write, record, or run scripts that trigger the same desktop actions that you trigger using the keyboard and mouse.

List $19.95 Our Price **$17.95** (BAFG)

SEE RELATED CATEGORY: **Scripting**

AppleScript Language Guide,

by Apple Computer, Inc.

- A complete reference for anyone using AppleScript to modify existing scripts or to write new ones.
- Contains useful information for programmers who are working on scriptable applications or complex scripts.
- Features detailed definitions of AppleScript terminology and syntax in the following categories: Value classes, commands, objects and references to objects, expressions, control statements, handlers, and script objects.
- Includes many sample scripts, discusses advanced topics such as writing command handlers for script applications, the scope of script variables and properties declared at different levels in a script, and inheritance and delegation among script objects.

List $29.95 Our Price **$26.95** (BALG)

SEE RELATED CATEGORY: **Scripting**

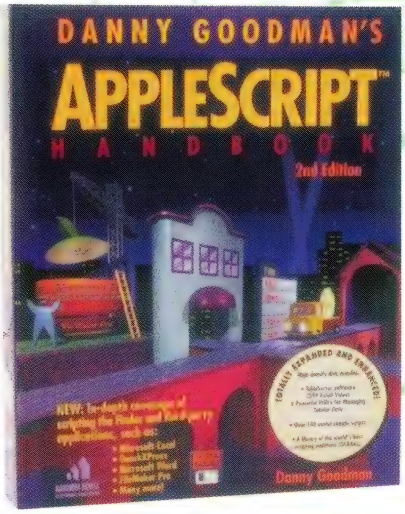
AppleScript Scripting Additions Guide

by Apple Computer, Inc.

- Use the standard scripting additions commands.
- Write scripting additions.

List $18.95 Our Price **$17.05** (BSCRADD)

SEE RELATED CATEGORY: **Scripting**



Danny Goodman's AppleScript Handbook Second Edition by Danny Goodman

- Customize and extend the capabilities of any Macintosh computer – no programming experience needed!
- Learn to use scripts to enhance the Macintosh environment, automate many processes, link data between applications, and much more.
- All-new examples showing how to integrate AppleScript with the Finder, spreadsheets, desktop publishing programs, graphics applications, databases, telecommunications programs, utilities, and HyperCard.
- Includes 3 1/2" disk with over $100 worth of software, including AppleScript 1.1, valuable utilities, and powerful, ready-to-use scripts.

List $39.00 Our Price **$35.00** (BDGASHB)

SEE RELATED CATEGORY: **Scripting**

SEE RELATED CATEGORY: **Tools, Libs & Utilities**

Includes Disk!

Applied Mac Scripting

- Learn to design and develop powerful scripts.
- Covers AppleScript™, Frontier, QuicKeys, Tempo II, nShell, FaceSpan Application Builder, Scripting PlainTalk and System 7.5.
- Hands on tutorial shows you how to automate your Macintosh activities by learning how to use the AppleScript and Frontier scripting environments.
- Harness the capabilities of a wide variety of Macintosh applications into the integrated productivity tools. This includes such things as the newspaper script which combines the power of SITcomm, MacWrite Pro, and FileMaker Pro, or QuarkXPress.

List $34.95 Our Price **$31.45** (BAPPLIED)

SEE RELATED CATEGORY: **Scripting**

PowerPC Programmer's Toolkit by Tom Thompson

- CD-ROM includes a special version of Metrowerks CodeWarrior 7.0 and sample code from the book
- Details how to write PowerPC applications in native code for blazing speed
- Written by an Apple Insider

List $45.00 Our Price **$40.50** (BPPCPT)

Mastering Netscape 4.0 for Macintosh, Second Edition by Greg Holden

- CD-ROM includes Netscape Plug-Ins and helper applications
- Uncovers hidden features of the program and how to best utilize them
- Web page for the book includes cool utilities and updated information

List $40.00 Our Price **$36.00** (BMASNET4)

Network Frontiers Bundle

by AP PROFESSIONAL

(Includes 3 books: Complete Guide to MAC,Backup Management, Designing AppleTalk Network Architectures, Managing AppleShare and Workgroup Servers)

- Apple Certified
- Each bundle includes the first 3 books that correspond to the Apple
- Certified Server Engineer (ACSE) program

List $89.95 Our Price **$59.95** (BNETFB)

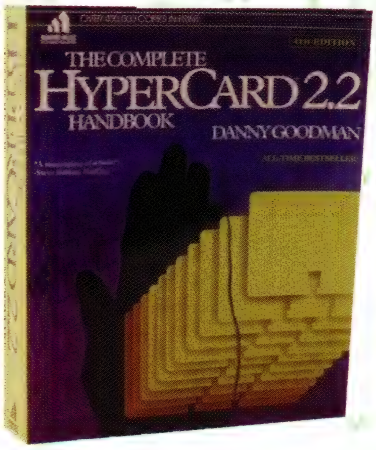
The Complete AppleScript Handbook by Danny Goodman

Master AppleScript with the definitive book/disk toolkit. This self-contained toolkit teaches you AppleScript from the ground up providing you with tools you need to automate tasks and integrate Macintosh applications. This book explains in detail all commands and usage of the AppleScript language including:

• Issuing command • Describing objects • Working with values, variables, and expressions • Using if-then constructions, loops and subroutines • Error checking and debugging • Scripting Finder-level processes • Using AppleScript with third party applications

List $35.00 Our Price **$31.50** (BAPLSCRHB)

SEE RELATED CATEGORY: **Scripting**



The Complete HyperCard® 2.2 Handbook Fourth Edition

by Danny Goodman

- The biggest-selling programming Mac book.
- Learn to build working applications using the latest version of HyperCard.
- Covers text, painting tools, extension commands (XCMDs), scripting in HyperTalk, and more.

List $35.00 Our Price **$31.50** (BHYPCRD2)

SEE RELATED CATEGORY: **Scripting**

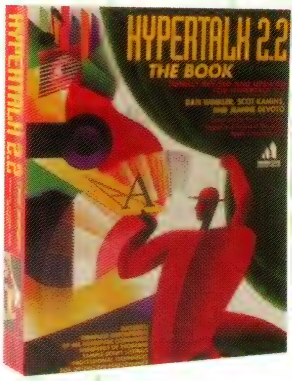
HyperTalk® 2.2: The Book Second Edition

by Dan Winkler, Scott Kamins, and Jeanne DeVoto

SEE RELATED CATEGORY: **Scripting**

- The most complete, authoritative source on HyperTalk 2.2 programming and troubleshooting.
- Covers each language element of HyperTalk 2.2 (including the odd quirk or bug).

List $35.00 Ours Price **$31.50** (BHYPTAL)



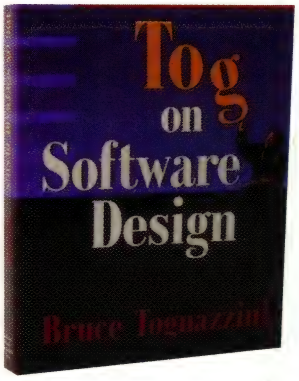
HyperCard Stack Design by Apple Computer, Inc.

HyperCard Stack Design Guidelines is an essential book for everyone who creates Apple® HyperCard stacks. Included are illustrated discussions of:

- Guidelines for stack development-audience evaluation, subject matter requirements and constraints, mode of presentation, and testing
- Navigation, the importance of making sure users can get around in your stacks
- Graphic Design and illustration- including the use of grids to determine card and background layout
- Text in stacks- placement, readability, and special considerations when writing for the screen
- Music and sound in stacks-as subject matter, reinforcement, entertainment, alert mechanism, or transition

List $21.95 Our Price **$19.95** (BHYPSTA)

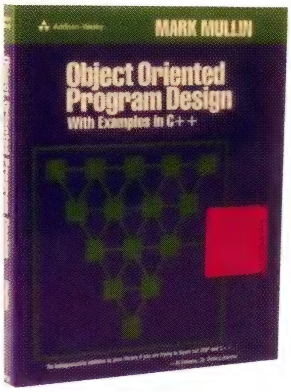
SEE RELATED CATEGORY: **Scripting**



Tog on Software Design by Bruce "Tog" Tognazzini

Respected industry futurist, Tog, presents his vision of our technological future, detailing the steps computer professionals need to take to deliver new technologies that will profit the industry and benefit society in general. Contains Tog's insights on a wide range of topics from quality management to the meaning of standards, and responses to queries supplied by designers and developers.

List $29.95 Our Price **$26.95** (BTOG)



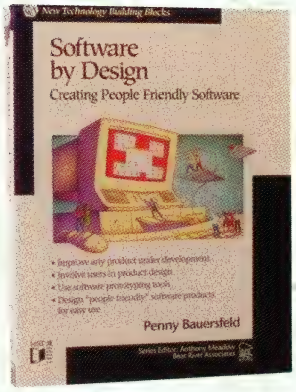
Object Oriented Program Design by Mark Mullin

- A concise guide to the essential concepts and techniques of OOP design
- Clarifies the key concepts of object oriented programming such as objects, classes, entities, hierarchies, and inheritance
- Uses typical database application to illustrate each OOP topic, to give the programmer a familiar point of reference

List $22.95 Our Price **$20.66** (BOOPRODES)

Software By Design: Creating User Friendly Software

by Penny Bauersfeld (Series Editor: Tony Meadow)



- A thorough how-to for designing software that is easy to learn and comfortable to operate.
- Written from the Macintosh perspective, but compatible with all platforms.
- Stresses user input from initial design, through prototyping, testing and revision.
- Provides tools for analyzing user needs and test responses, plus exercises for sharpening user-oriented design skills.

List $29.95 Our Price **$26.95** (BDESIGN)

Inside Macintosh®: CD-ROM by Apple Computer, Inc.

- More than 25 volumes in electronic form.
- Includes: QuickDraw™ GX Library, Macintosh Human Interface Guidelines, PowerPC System Software, Macintosh Toolbox Essentials and More Macintosh Toolbox, QuickTime and QuickTime Components.
- Access over 16,000 pages of information with Hypertext linking and extensive cross referencing.

List $99.95 Our Price **$89.95** (BIMCD)

Inside Macintosh®: Sound by Apple Computer, Inc.

- Describes the parts of the Macintosh system software that allow you to manage sounds.
- Contains information to write applications that can record and play back sounds, compress and expand audio data, convert text to speech, and perform other similar operations.

List $26.95 Our Price **$14.98** (BIMSOUND)

Inside Macintosh®: Imaging by Apple Computer, Inc.

- Covers QuickDraw and Color QuickDraw.
- Includes general discussions of drawing and working with color.
- Describes the structures that hold images and image information, and the routines that manipulate them.
- Covers the Palette, Color, and Printing Managers, and the Color Picker, Color Matching, and Picture Utilities.

List$26.95 OurPrice **$16.48** (BIMIMAG)

Inside Macintosh®: QuickTime by Apple Computer, Inc.

- For developers who want to create applications that use QuickTime, the system software that allows the integration of video, animation, and sounds into applications.
- Describes all of the QuickTime Toolbox utilities.
- Provides the information you need to compress and decompress images and image sequences.

List $29.95 Our Price **$14.98** (BIMQT)

Inside Macintosh®: QuickTime Components by Apple Computer, Inc.

Covers how to use and develop QuickTime components such as image compressors, movie controllers, sequence grabbers, and video digitizers.

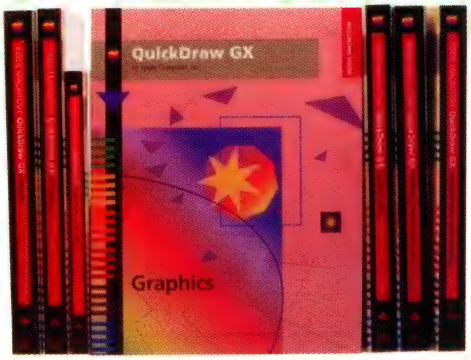
List $34.95 Our Price **$17.47** (BIMQTCOM)

One & 2-Day SHIPPING (call for details)

Inside Macintosh®: QuickDraw™ GX Programmer's Overview

- Provides an introduction to QuickDraw™ GX, providing an overview of the QuickDraw GX environment from a developer's perspective.
- Introduces the QuickDraw™ GX programming and runtime environments, the relationship between QuickDraw GX and the rest of the Macintosh® systems software and the relationship between QuickDraw GX and Macintosh applications.
- Learn the key elements of QuickDraw GX programming, data structures, object types, and functions used most frequently by QuickDraw GX developers are also covered.
- Provides a series of practical examples demonstrating how to approach programming with QuickDraw GX.

List $24.95 Our Price **$12.48** (BIMGXOV)



Inside Macintosh®: QuickDraw™ GX Graphics
by Apple Computer, Inc.

- Shows how to create and manipulate the fundamental geometric shapes of QuickDraw GX to generate a vast range of graphic entities.
- Demonstrates how to work with bitmaps and pictures, and specialized QuickDraw GX graphic shapes.

List $26.95 Our Price **$15.98** (BIMGXGR)

Inside Macintosh®: QuickDraw™ GX Objects
by Apple Computer, Inc.

Introduces QuickDraw GX and its object structure, and shows programmers how to manipulate objects in all types of programs.

List $26.95 Our Price **$15.98** (BIMGXOBJ)

Inside Macintosh®: QuickDraw™ GX Printing by Apple Computer, Inc.

- Essential for any developer whose QuickDraw™ GX application supports printing.
- Shows how to support the new printing features of QuickDraw GX, including desktop printers and expandable printing dialog boxes.
- Shows how to use printing-related objects to add custom panels to printing dialog boxes and to create custom page formats.

List $26.95 Our Price **$14.98** (BIMGXPRNT)

Inside Macintosh®: QuickDraw™ GX Printing Extensions and Drivers by Apple Computer, Inc.

- Essential for developers who want to create extensions to the application printing capabilities of QuickDraw™ GX, or who need to write a printing device driver that works with QuickDraw GX.
- Describes how to create printing extensions and printer drivers, and provides a complete reference to the messages, functions, and resources that they use.

List $29.95 Our Price **$14.98** (BIMGXEXT)

Inside Macintosh®: QuickDraw™ GX Typography by Apple Computer, Inc.

- Essential for developers who use QuickDraw™ GX to manipulate text.
- Shows how to use QuickDraw GX objects to handle all kinds of text – from plain, unstyled text to complex, mixed-direction and multi-language text with sophisticated stylistic and typographic variations.
- Shows how to create and manipulate the three different types of text shapes supported by QuickDraw GX including text shapes, glyph shapes, and layout shapes.

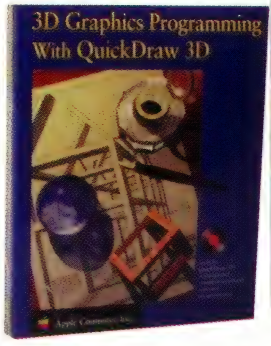
List $29.95 Our Price **$14.98** (BIMGXTYP)

Inside Macintosh®: QuickDraw™ GX Environment and Utilities by Apple Computer, Inc.

- Companion to QuickDraw™ GX Objects.
- Contains programming information useful to any developer writing QuickDraw GX applications.
- Describes QuickDraw GX memory management, error handling, debugging, and mathematical functions, as well as conversion from QuickDraw to QuickDraw GX.

List $29.95 Our Price **$15.98** (BIMGXENV)

3D Graphics Programming Using QuickDraw 3D
by Apple Computer, Inc.



- Incorporate spectacular 3D graphics into your applications.
- Explore QuickDraw 3D, a revolutionary graphics extension to the Mac OS for Power Macintoshes.
- CD contains the complete QuickDraw 3D system itself and a complete database of the QuickDraw 3D API, allowing you instant access to the hundreds of graphics calls via a fast viewing engine. Book/CD-ROM, 640 pages.

List $39.95 Our Price **$35.96** (B3DGRAP)

3D Game Machine v1.2 by Virtually Unlimited

- Create lightning-fast 3D arcade games and interactive multimedia applications.
- Ultra-fast rendering – 15 frames per second on a 14" monitor completely texture-mapped, with a PowerMac 6100/60
- Create full "virtual" 3D worlds with six degrees of freedom, free-form texture mapping, shading, material and light properties, convex/cave polygons with unlimited vertices, unlimited light sources, dynamic hidden surface removal, special graphic modes for fast full-screen animation, collision detection, explosion simulation, 3D data importing.
- Simple easy-to-use interface. Runs on all Macs! Works with CodeWarrior.

Our Price **$299.00** + license. (S3DGAME)

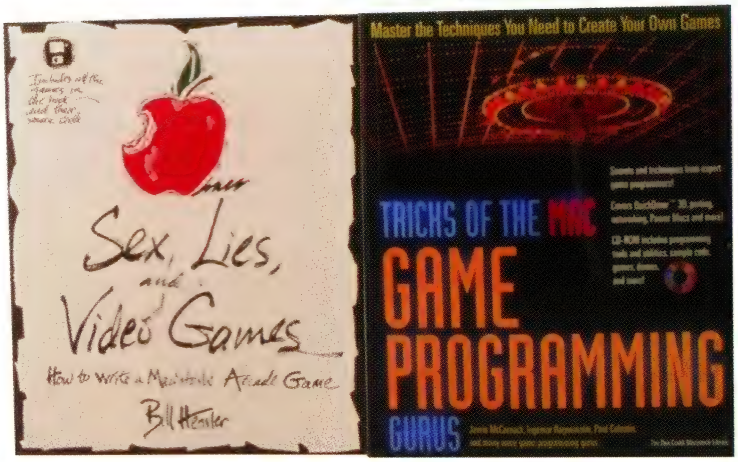
Sex, Lies and Video Games by Bill Hensler

- A learn-by-example tutorial on the ins and outs of Mac arcade-style game programming in C.
- Features game theory, sprite animation, sound, and interaction techniques.
- A must-read for serious programmer's and hobbyists alike.

List 34.95 Our Price **$31.46** (BSEX)

SEE RELATED CATEGORY:
Tools, Libs & Utilities

Includes CD-ROM!



Tricks of The Mac Game Programming Gurus

- For beginning to expert game programmers
- Complete overview of all the necessary components of game programming on the Macintosh.
- Packed with valuable tools, utilities, sample code, CodeWarrior™ Lite and game demos.
- QuickDraw 3D and Power Mac optimization and inside info on how Glypha III was created.
- Hundreds of tried-and-true tricks, tips, and insider secrets from well-known Mac game programming experts

List $50.00 Our Price **$45.00** (BTRICKS)

Black Art of Macintosh Game Programming:

by; Kevin Tieskoetter.

- Develop your own 3D games in C on the Mac.
- Includes CD with project files for both Symantec C and Code Warrior
- Create freeform texture-mapped games and polygon graphics
- Control dynamic source code-all compatible as native to the Power Mac
- Write directly to the screen, bypassing QuickDraw

List $39.99 Our price **$35.99** (BBLACK)

Includes CD-ROM!

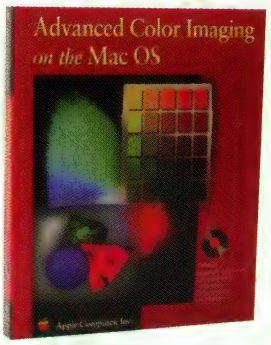
Graphic Gems V Edited by Alan W. Paeth

- Loaded with practical tools for implementing new ideas and techniques, to offer working solutions to real programming problems.
- Contains over 40 new gems in ellipses, splines, Bezier curves, and ray tracing – displaying the most recent and innovative techniques in graphics programming.
- Includes a disk with source code from all five volumes. Available in both IBM and Macintosh versions. CONTENTS: Algebra and Arithmetic. Computational Geometry. Modeling and Transformation. Curves and Surfaces. Ray Tracing and Radiosity. Halftoning and Image Processing. Utilities.

List $49.95 Our Price **$44.95** (BGEMS5)

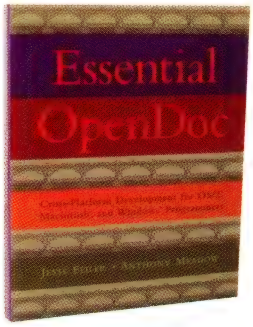
Includes Disk!

Advanced Color Imaging on the Mac OS



- Enhance your software's color capabilities with step-by-step instructions.
- Augment the color support supplied with QuickDraw, and QuickDraw GX.
- Use the Pallette Manager to get the best colors on limited displays.
- Match colors between screens and input/output devices (scanners & printers)
- CD includes a complete reference information in both QuickView and Acrobat formats. Plus, a sample application demonstrating ColorSync programming techniques.

List $36.95 Our Price **$33.25** (BADVCI)



OpenDoc Programmer's Cookbook

- Shows you how to create OpenDoc software components, called parts editors, for the Mac OS Platform.
- Including instructions for setting up the Macintosh Programmers Workshop (MPW) development environment to write OpenDoc software
- Annotated listings of explaining the methods that implement the SamplePart part editor
- Descriptions of other sample part editors created by the OpenDoc engineering team to illustrate more advanced features
- Summary descriptions of software utilities provided with OpenDoc for the Mac OS
- An Introduction to the System Object Model (SOM) technology underlying OpenDoc

List $24.95 Our price **$22.45** (BODCOOK)

Essential OpenDoc

- Gives an in-depth look at the technical issues of OpenDoc
- Explores the three core technologies that support it's functionality – SOM, OpenDoc's storage mechanism, and the Open Scripting Architecture (OSA).
- Also examines CyberDog, a set of OpenDoc part editors that provides access to Internet services and offers compelling example of the power of OpenDoc development

List $39.95 Our price **$35.95** (BESOD)

Inside CodeWarrior 9

- Includes CodeWarrior IDE User's Guide.
- This is the printed version of the documentation provided on the CD.
- Covers CodeWarrior, the debugger, and associated tools.

Our Price **$34.95** (BINSCW)

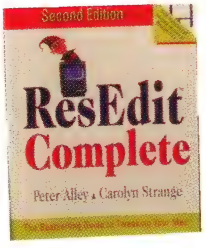
SEE RELATED CATEGORY:
Dev. Environments

ResEdit™ Complete, Second Edition

by Peter Alley and Carolyn Strange

- Customize every aspect of your interface form creating screen backgrounds and icons to customizing menus and dialog boxes. 608 pages. Book/disk package.

List $34.95 Our Price **$31.45** (BRESED2)



Inside PowerPlant Manual

- Create PowerPlant applications using the CodeWarrior IDE and PowerPlant Constructor.
- Full descriptions of major PowerPlant classes and resources.
- Included are the PowerPlant Constructor Manual, including View, TextTraits and Custom Types editing, and PowerPlant Library Reference, covering all classes and functions in PowerPlant.

Our Price **$34.95** (BINSPP)

SEE RELATED CATEGORY:
Dev. Environments

OpenDoc Programmer's Guide by Apple Computer, Inc.

- The official reference for the implementation of OpenDoc on the Mac OS.
- Describes the component software revolution and explains how to develop for it on the Mac OS platform.
- Accompanying CD-ROM contains a complete reference to the OpenDoc programming interface, and an extensive collection of tested, reusable sample code.

List $44.95 Our Price **$40.46** (BOPENDOC)

C++ Programming with CodeWarrior by Jan L. Harrington

- Beginning OOP for the Macintosh and Power Macintosh and Mac OS compatibles.
- Learn object-oriented programming techniques using C++ as the example language and Metrowerks and CodeWarrior as the example compiler.
- Enclosed CD contains example code from the book and a full-function Metrowerks CodeWarrior

List $35.95 Our Price **$32.35** (BCPPCW)

The ResEdit All Night Diner by David Ciskowski

- An idea-filled menu and introduction to the joys of customizing software.
- Add personality to the Mac by customizing default icons, the text of menus and dialog boxes, cursors, pointers and more.
- Disk features ResEdit, plus lots of sample resources

Includes Disk!

List $24.95 Our Price **$22.45** (BRESDINE)

SEE RELATED CATEGORY: Dev. Environments

C++ Programming W/MacApp by David Wilson, Larry Rosenstein & Dan Shafer

- Learn the secrets to unlocking the power of MacApp®, Apple's development environment for C++
- Learn to design complex windows and views using the ViewEdit tool
- Learn to support multipage text and graphics with only five lines of code
- Learn to support Undo for menu commands and drawing operations that use the mouse.

List $34.95 Our Price **$31.46** (BCPPMACAP)

SEE RELATED CATEGORY: Tools, Libs & Utilities

Programming in Symantec C++ for the Macintosh by Judy May and John Whittle

- An introduction to object-oriented programming, the C++ language, and Symantec C++ for the Macintosh.
- Great for both programmers and beginners alike.
- Covers everything from the basics to advanced features of Symantec C++.
- Includes helpful examples of C++ code that illustrate object-oriented programs.

List $29.95 Our Price **$26.95** (BPSYMCPP)

SEE RELATED CATEGORY: Dev. Environments

Symantec C++ Programming by Neil Rhodes & Julie McKeehan

Symantec C++ Programming for the Macintosh is a tutorial for getting up and running in the Symantec C++ environment, while mastering the techniques of object-oriented programming.

- Explore the Symantec C++ environment, from debugging a program and using resource utilities to building applications and creating objects
- Design programs for compatibility with multiple application frameworks
- Learn how to use the new Visual Architect, Inspector, and templates.

List $45.00 Our Price **$40.50** (BSYMCPP)

SEE RELATED CATEGORY: Dev. Environments

Metrowerks CodeWarrior Programming by Dan Parks Sydow

- Includes CodeWarrior Lite, and Full Coverage of PowerPlant™.
- The best information on Metrowerks CodeWarrior, giving full coverage to the Gold Edition.
- CD includes Code Warrior Lite.

Includes CD-ROM!

List $39.95 Our Price **$35.95** (BCWPROG)

SEE RELATED CATEGORY: Dev. Environments

Dan Shafer Presents the Power of Prograph CPX

- Master the revolutionary graphical object-oriented programming language.
- Step by step course through three interrelated projects of increasing complexity.
- Learn Prograph language, CPX classes and object editors.
- Includes disk with all code in the book.

List $49.95 Our Price **$19.95** (BDANPRO)

Order Toll-free
800-MACDEV-1
(800-622-3381)

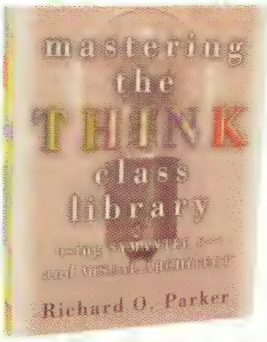
Visual Programming with Prograph CPX

by Scott B. Steinman and Kevin G. Carver

- An introduction to the language and a guide for advanced users, for both Macintosh and Windows-based machines.

List $34.00 Our Price **$30.60** (BVISPRO)

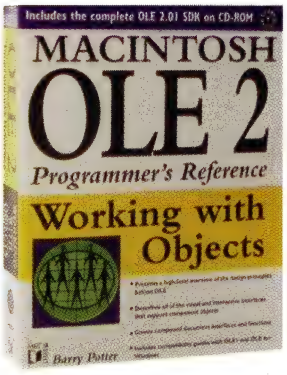
Mastering the THINK Class Library by Richard Parker



- Provides a thorough examination of Symantec's extensive Class Library and the Visual Architect.
- A complete description of the structure and operation of the TCL includes explanations of all code generated by the Visual Architect, any necessary custom code, and the operation of this code.
- Visual Architect tutorials provide you with a step-by-step approach for simplifying the development of complex Macintosh applications. 496 pages.

List $29.95 Our Price **$26.95** (BMASTERTCL)

SEE RELATED CATEGORY: **Dev. Environments**



Macintosh OLE2 Programmer's Reference: Working with Objects

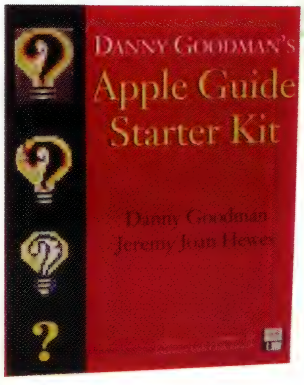
- Complete reference to the extensible protocol of Object Linking and Embedding, version 2.01 for Macintosh System 7.
- Describes the visual and interactice interfaces that support the component objects.
- Provides details of the OLE 2.01 for the Macintosh user Interface, addresses the issues of object class registration, shows how to implement the drag and drop objects from one application to another, covers the interface that exposes the basic embedding functionality, includes descriptions of API functions and more!

Includes CD-ROM!

List $44.95 Our Price **$40.45** (BOLE2)

Danny Goodman's Apple Guide Starter Kit

by Danny Goodman and Jeremy Joan Hewes.



- Create your own Apple Guide databases quickly and easily, without having to learn a scripting language, write coded files, or use several different files and programs
- Includes advice and tips on how to design a good Guide, from planning and creation through testing, revising, and indexing. Book/disk, 320 pages.

List $34.95 Our Price **$31.46** (BDGAGSK)

MacsBug Reference & Debugging Guide For MacsBug version 6.2

by Apple Computer, Inc.

- MacsBug is an assembly-language-level debugging tool
- Macros, templates, dcmds, and other resources for making debugging easier
- Macintosh memory management and the operating system as they relate to low level debugging
- How to display and set memory and process registers
- Disassemble memory, and set execution breakpoints
- Discipline, a tool for testing the validity of toolbox parameters
- Debugging strategies you can use to find and cure common bugs
- Includes MacsBug 6.2.

List $34.95 Our Price **$31.46** (BBUGREF)

AppleGuide Complete

by Apple Computer, Inc.

Includes CD-ROM!

- Covers Guide Maker, the software you use to build and test guide files.
- Learn about the complete cycle of designing as well as advanced topics such as scripting and coding guide files. Book/CD-ROM, 544 pages.

List $39.95 Our Price **$35.96** (BAPLGD)

SEE RELATED CATEGORY: **Tools, Libs & Utilities**

Programming For The Newton: Software Development using NewtonScript

by Julie McKeehan and Neil Rhodes. Foreword by Walter R. Smith

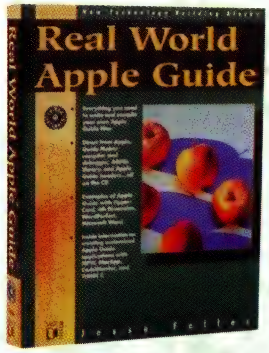
- An indispensable tool for Newton programmers.
- Includes disk with sample Newton application from the books, as well as demonstration version of Newton Toolkit (NTK) – the complete development environment for the Newton®.
- A Publication of AP Professional May 1994, Paperback, 393 pp.

List $29.95 Our Price **$26.95** (BPROGNEWT)

SEE RELATED CATEGORY: **Dev. Environments**

Lowest price guaranteed. Feel Good about your purchase! 30-day return policy.

Real World AppleGuide For The Mac

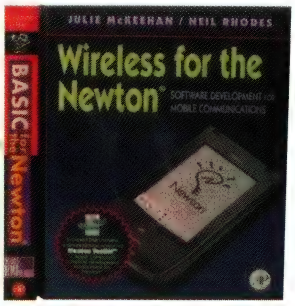


- An introduction to AppleGuide for programmers. It explains its design and function, plus how to design your own guides using AppleScript.
- Includes a disk of sample AppleGuides for AppleGuide-compliant applications.

Includes Disk!

List $39.95 Our Price **$35.95** (BREALWLD)

SEE RELATED CATEGORY: **Tools, Libs & Utilities**



Wireless For The Newton: Software Development for Mobile Communications

by Julie McKeehan and Neil Rhodes

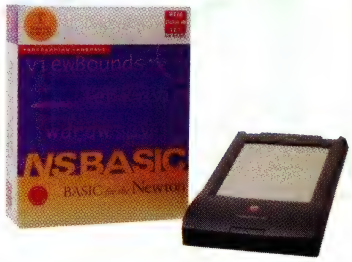
- Learn to develop Newton® software on the Macintosh.
- Hands-on Newton environment training with sample code
- Includes disk with sample source code for a Newton application, as well as demonstration NTK™ – the complete development environment for the Newton®.

List $34.95 Our Price **$31.45** (BWIRELESS)

Programming for the Newton Using NS BASIC

by John Schettino & Liz O'Hara

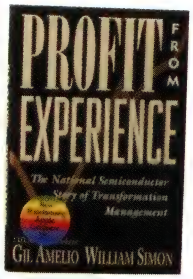
Includes Disk!



- Program on Macintosh, Windows-based PC, or on the Newton itself.
- Straight-forward "programming by example" approach – you'll be writing Newton programs right away.
- Includes 3.5" disk containing Demonstration NS BASIC and over fifty example programs. (Newton not included)

List $99.00 Our Price **$94.99** (SNSBASIC)

SEE RELATED CATEGORY: **Dev. Environments**



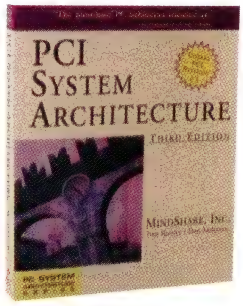
Profit From Experience by Gil Amelio and William Simon

Written by the new CEO of Apple

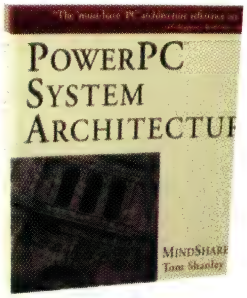
- The story of the transformation of National Semiconductor – how Amelio and his management team took it from it's worst loss in 30 years to the highest earnings in it's history • Includes: 6 core business issues and why they are critical to success • The TEAM program • 10 personal attributes to strive for • Amelio's guidelines for General Managers • Attributes to look for when hiring.

List $24.99 Our price **$22.45** (BPROFIT)

PCI System Architecture, Third Edition by MindShare



Describing revision 2.1 of the Peripheral Component Interconnect (PCI) bus specification, this book explores PCI's relationship to the rest of the system. It includes an in-depth treatment of PCI to PCI bridges, the PCI BIOS, the 66MHz PCI bus, and more. 592 pages. List $34.95 Our Price **$31.46** (BPCISYS)



PowerPC System Architecture

by MindShare Inc. and Tom Shanley

- Describes the hardware architecture of the PowerPC systems, providing clear, concise explanation of the PowerPC specification, the template upon which all PowerPC processors are designed.
- Includes the specs for both the 32 and 64 bit implementations including • supervisor privilege level facilities • logical memory addresses • I/O and memory mapped I/O • address translation for segments, pages, and blocks • virtual paging • interrupts.

List $34.95 Our Price **$31.46** (BPPCARCH)

Inside Macintosh®: AOCE Application Interfaces

by Apple Computer, Inc.

- Shows how your application can take advantage of the system software features provided by PowerTalk system software and the PowerShare collaboration servers.
- Add electronic mail capabilities to your application, write a messaging application or agent, store information in and retrieve information from PowerShare and other AOCE catalogs.
- Add catalog-browsing and find-in-catalog capabilities to your application, write templates that extend the Finder's ability to display information in PowerShare and other AOCE catalogs.
- Add digital signatures to files or to any portion of a document, and establish an authenticated messaging connection.

List $40.45 Our Price **$22.48** (BIMAOCE)

Inside Macintosh®: Interapplication Communication

by Apple Computer, Inc.

Shows how applications can work together. How your application can share data, request information or services, allow the user to automate tasks, communicate with remote databases.

List $34.95 Our Price **$18.48** (BIMIAPP)

Inside the Macintosh Communications Toolbox

by Apple Computer, Inc.

- The definitive reference to the Macintosh Communications Toolbox, an integral part of the System 7 Macintosh Toolbox that enables developers to create communications applications or add communications features to other applications.
- Describes all of the routines that provide programmers with standard access to important communications services and in addition enables programmers to extend the reach of the Macintosh into non-Apple environments.

List $24.95 Our Price **$22.45** (BCOMM)

Inside Macintosh®: Networking

by Apple Computer, Inc.

- Describes how to write software that uses AppleTalk networking protocols.
- Describes the components and organization of AppleTalk and how to select an AppleTalk protocol.
- Provides the complete application interfaces to all AppleTalk protocols, including ATP (AppleTalk Transaction Protocol), DDP (Datagram Delivery Protocol), and ADSP (AppleTalk Data Stream Protocol), among others.

List $29.95 Our Price **$14.98** (BIMNET)

Inside Macintosh®: AOCE Service Access Modules by Apple Computer, Inc.

- Describes how to write a software module that gives users and PowerTalk-enabled applications access to a new or existing mail and messaging service or catalog service.
- Shows how to write a catalog service access module (CSAM), a messaging service access module (MSAM), and AOCE templates that allow a user to set up a CSAM or MSAM and add addresses to mail and messages.

List $26.95 Our Price **$14.98** (BIMAOCES)

Inside Macintosh®: Text by Apple Computer, Inc.

- Describes how to perform text handling, from simple character display to multi-language processing.
- Covers Font, Script, Text Services, and Dictionary Managers, in addition to QuickDraw Text, TextEdit, and International and Keyboard Resources.

List $39.95 Our Price **$19.98** (BIMTEXT)

Inside Macintosh®: Devices by Apple Computer, Inc.

- Describes how to write software that interacts with built-in and peripheral hardware devices.
- Learn how to write and install your own device drivers, desk accessories, and Chooser extensions.
- Communicate with device drivers using the Device Manager; access expansion cards using the Slot Manager; control SCSI devices using SCSI Manager 4.3 or the original SCSI Manager.
- Communicate directly with Apple Desktop Bus devices; interact with the Power Manager in battery-powered Macintosh computers, and communicate with serial devices using the Serial Driver.

List $29.95 Our Price **$14.98** (BIMDEV)

Inside Macintosh®: X-Ref by Apple Computer, Inc.

Is a fast access to all the information in Inside Macintosh. Inside Macintosh X Ref; Provides programmers with a quick and easy way to find the exact information they need in this definitive suite of books, (all 26 volumes). It is indexed by topic, volume, chapter, and accompanying page number.

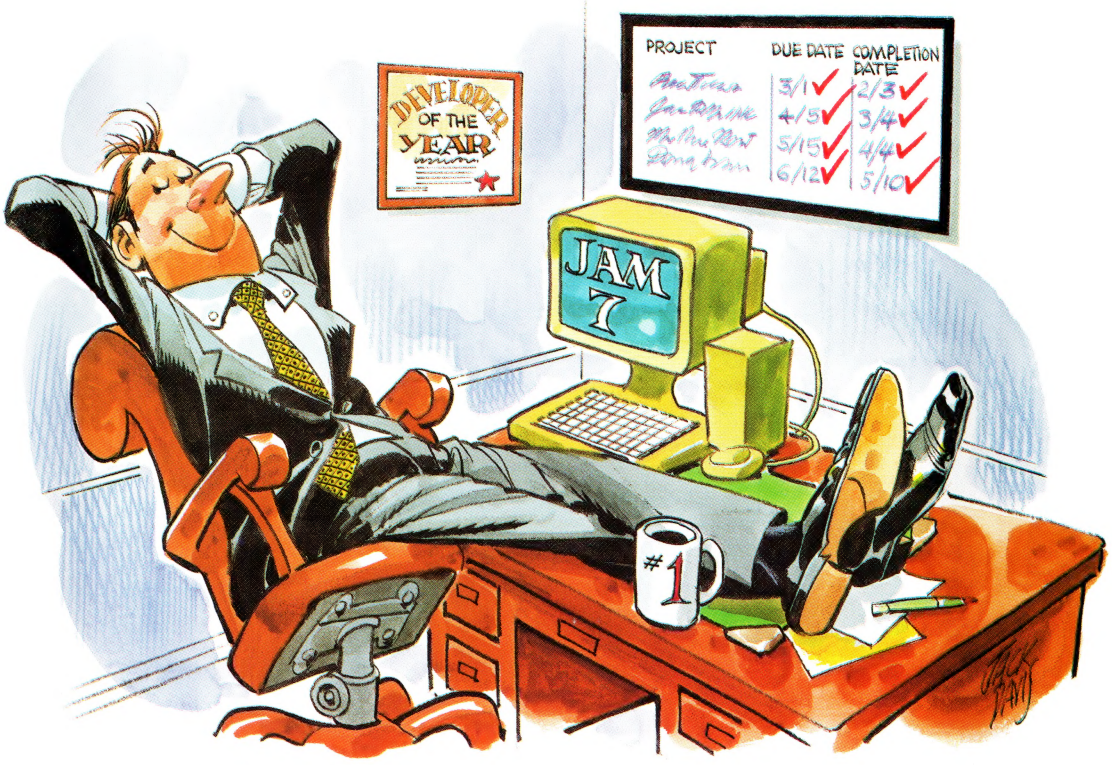
List $19.95 Our Price **$9.98** (BIMXREF)

Order Toll-free
800-MACDEV-1
(800-622-3381)

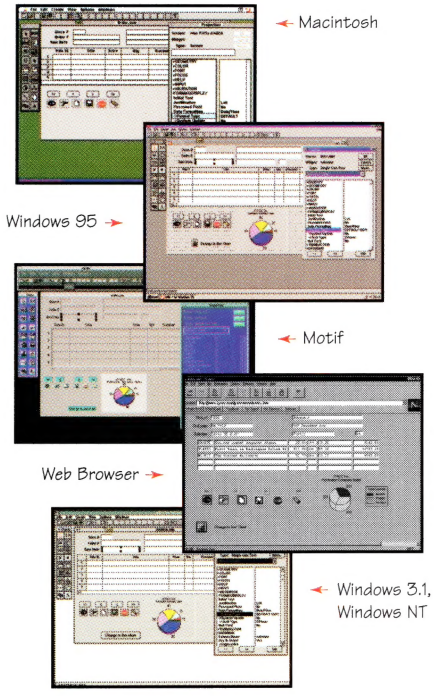
All entries in this index are alphabetized.
For your convenience the product names are **bold**, book names are *italicized* and company names are in plain text.

INDEX

Life is Sweet with JAM!



Get JAM and enjoy success in building and deploying client/server and Web applications



In a world where requirements change and deadlines don't, only JAM gives you what you need to build powerful, completely portable client/server, 3-tier and Web applications. JAM's cross-platform RAD capabilities enable you to successfully develop and deploy demanding applications across any platform, database, GUI or Web Browser.

Are you willing to base your project's success on a tool that only *promises* portability? JAM lets you build great looking applications that are fully portable *today*. If you need to build serious client/server, 3-tier or Web applications, call JYACC for a free demonstration kit or white paper.

- ✓ Desktop ease with the power of a 2nd-generation tool
- ✓ Unrivaled portability between Windows® 3.1, NT, 95; Macintosh®; OS/2 Warp®; Character, Motif and Web Browsers
- ✓ Unparalleled database access to Oracle®, Sybase®, Informix®, ODBC and more
- ✓ Automatic SQL generation with full transaction control
- ✓ Visual repository-driven development of both client and application server
- ✓ Centralized control over application objects via inheritance
- ✓ Unique architecture for team development
- ✓ No runtimes

Find out for yourself how sweet life can be with JAM!

Call 1 800 458-3313.

Or E-mail: sweetlife3@jyacc.com for a free demonstration kit or white paper.
Visit our Web site at http://www.jyacc.com
For international inquiries call: 1-212-267-7722 or FAX 1-212-608-6753

JYACC
Meeting the needs of professional developers for over 18 years

BRAZIL (55) 11 816 6228 • FINLAND (358) 3486 4000 • FRANCE (33) 1 46 92 45 44 • GERMANY (49) 40 79 70 07 0 • HRVATSKA (385) 1 242 116 • ISRAEL (972) 3 557 3675
ITALY (39) 2 45 2701 • MEXICO (52) 5 663 0405 • RUSSIA (7) 095 288 1924 • SAUDI ARABIA (966) 2 665 8406 • SINGAPORE (65) 220 8322 • SLOVENIA (386) 61 1405 004
SPAIN (34) 1 804 0625 • SWEDEN (46) 8 15 10 10 • SWITZERLAND (41) 21 991 9041 • THAILAND (66) 2 513 3559 • THE NETHERLANDS (31) 70 320 9214 • UNITED KINGDOM (44) 171 814 6660